

Alberto E. Gabás Royo

SOLAR IRRADIANCE FORECASTING USING NEURAL NETWORKS

Faculty of Information Technology and Communication Sciences (ITC)
Master of Science Thesis
September 2019

ABSTRACT

Alberto E. Gabás Royo: Solar Irradiance Forecasting Using Neural Networks
Master of Science Thesis
Tampere University
Industrial Engineering
September 2019

Accurate solar irradiance forecasting is essential for minimizing operational costs of solar photovoltaic (PV) generation as it is commonly used to predict the power output. This thesis presents and compares three different machine learning approaches of solar irradiance forecasting: Random Forest (RF), Feedforward Neural Networks (FNNs) and Long Short-Term Memory (LSTM) networks. Each model was tested on two different forecasts: the next hour average and the hourly day-ahead averages. The machine learning algorithms were trained and tested on data from a weather station located at Tampere University (TAU) in Tampere, Finland. Data were pre-processed before training the algorithms and the relevant features were selected. Moreover, Grid Search and Random Search techniques were used along with multiple train and validation splits to find the optimal hyperparameters for each machine learning algorithm. Persistence model is set as a baseline model for comparison while RMSE and MAE are used to quantify the prediction error. For the next hour forecast, LSTM achieved the highest accuracy in terms of RMSE (76.14 W/m²), 2.1% and 1.1% better than RF and FNN respectively. Instead, FNN generally produced the best results in the day-ahead forecast. In all models, the prediction error increases as the forecast horizon increases until it stabilizes at 10 hours approximately. Further, the error keeps increasing but slower. Besides, the next hour forecast models were able to predict considerably better the next hour solar irradiance than the day-ahead forecast models.

Keywords: solar irradiance, neural networks, forecasting, prediction, machine learning

The originality of this thesis has been checked using the Turnitin OriginalityCheck service.

PREFACE

This thesis is made as a completion of my Master in Industrial Engineering with Energy specialization. The research was undertaken at Tampere University in Tampere, Finland during my Erasmus program.

I had previously worked with neural networks in my graduate thesis and found the field of artificial intelligence, which is currently growing, very interesting. Therefore, I was very glad when I was offered the topic of this research for my master's thesis as it merges two fields of great interest to me: renewable energy and neural networks.

I would like to thank first my supervisor Seppo Valkealahti for his guidance and support during this process. I also wish to thank Kari Lappalainen for his help during the data importation process and Heikki Huttunen for his advise about machine learning and neural networks. Last, I do not want to forget to thank my Erasmus fellows as some good ideas for my research came up from conversations with them.

It was a long journey with good moments when results showed easily and moments when they seemed to be so far away despite my laptop running for countless nights. Nonetheless, I enjoyed the whole process and I hope you enjoy this lecture the same way.

Tampere, 2nd September 2019

Alberto E. Gabás Royo

CONTENTS

1	Introduction	1
2	Theoretical Background	2
2.1	Solar Irradiance	2
2.2	Machine Learning	3
2.2.1	Supervised Learning	3
2.2.2	Random Forests	4
2.2.3	Artificial Neural Networks	8
2.2.4	Long Short-Term Memory Networks	11
2.3	Time Series Data	13
2.3.1	Baseline Predictions	13
2.3.2	Multiple Train and Validation Splits	13
2.4	Hyperparameter Optimization	15
2.5	Performance Metrics	16
2.6	Used Software	16
3	Research Methodology	17
3.1	Weather Data and Sensors	17
3.2	Data Preprocessing	18
3.3	Train and Validation Data	23
3.4	Persistence Algorithm	24
3.5	Random Forest Experiments	24
3.5.1	Variable selection	24
3.5.2	Next Hour Forecast	27
3.5.3	Day Ahead Forecast	30
3.6	Feed-Forward Neural Network Experiments	34
3.6.1	Next Hour Forecast	34
3.6.2	Day Ahead Forecast	40
3.7	Long Short-Term Memory Network Experiments	45
3.7.1	Data structure	45
3.7.2	Next Hour Forecast	46
3.7.3	Day Ahead Forecast	54
4	Results and Discussion	61
5	Conclusions	69
	References	70
	Appendix A Variable importances	72

LIST OF SYMBOLS

b_i	bias term of neuron i
b_C	input modulation gate bias
b_f	forget gate bias
b_i	input gate bias
b_o	output gate bias
C_t	cell state at time t
\hat{C}_t	new cell state candidates at time t
c_w	weight constraint
d	day of the year
d_{cos}	cosinusoidal transformation of day of the year
d_{sin}	sinusoidal transformation of day of the year
f_a	activation function
f_t	forget gate output
G	global horizontal irradiance
G_b	direct normal irradiance
G_d	diffuse horizontal irradiance
h_t	hidden state at time t
$h_{W,b}$	output of a Neural Network neuron
i_t	input gate output
n	number of observations
N_d	number of days in the year
N_{tf}	number of time fractions in a day
o_t	output gate output
t	time of the day
t_{cos}	cosinusoidal transformation of time of the day
t_{sin}	sinusoidal transformation of time of the day
\vec{w}	weight vector of a neuron
W_C	input modulation gate weights
W_f	forget gate weights

$W_{i,j}$	weight j of neuron i
W_i	input gate weights
W_o	output gate weights
X	input vector
x_j	input j
x_t	input at time t
\hat{y}_i	predicted observation
y_i	actual observation
z	feature value of one sample
z_{max}	maximum value of a feature in dataset
z_{min}	minimum value of a feature in dataset
z_{norm}	feature value of one sample normalized
θ_Z	zenith angle
σ	sigmoid function

LIST OF ABBREVIATIONS

ANN	Artificial Neural Network
DHI	Diffuse Horizontal Irradiance
DNI	Direct Normal Irradiance
FNN	Feedforward Neural Network
GHI	Global Horizontal Irradiance
IEA	International Energy Agency
LSTM	Long Short-Term Memory
MAE	Mean Absolute Error
MSE	Mean Squared Error
NaN	Not a Number
PV	photovoltaic
ReLU	Rectified Linear Unit
RF	Random Forest
RMSE	Root Mean Squared Error
RMSprop	Root Mean Square prop
RNN	Recurrent Neural Network
SGD	Scaled Gradient Descent
SI	international system of units (Système international d'unités in French)
SI	Solar Irradiance
TAU	Tampere University

1 INTRODUCTION

Global renewable electricity capacity is expected to grow by over 1 TW, a 46% growth over the period 2018 to 2023, according to the International Energy Agency (IEA) [14]. Solar PV represents more than half of this expansion and dominates the renewable capacity growth. 575 GW of new PV capacity are expected to become operational over the period 2018 to 2023, which represents a 143% growth where utility-scale projects account for 55% of this expansion. The yearly growth is expected to accelerate after 2020 with increasing cost-competitiveness and continuous policy support. Consequently, global net solar PV capacity additions should exceed 110 GW per year by 2023.

Solar PV generates power from sunlight transforming the solar irradiance into power. The performance of PV systems is affected by weather parameters like temperature. Thus, PV power depends on the solar irradiance and meteorological conditions, which add variability and uncertainty to the PV generation.

Considering the current scenario of solar PV growth, the modern grid faces the challenge of the mentioned uncertainty in power generation. There are solutions to deal with this challenge as energy storage to compensate the intermittency of PV generation [12]. Furthermore, knowing how much PV power will be produced could considerably reduce the operational costs of power plants. Thus, accurate PV power forecasting for different timescales (weekly, day-ahead, next hour and intra-hour) is critical for effectively integrating solar energy into the grid [10].

As PV power output is strongly dependent on solar irradiance, solar irradiance forecasting has been broadly studied in the literature. The forecasting methods could be split into three categories: physical, statistical and machine learning methods [12]. Physical models make predictions based on physical laws that govern the weather [10]. Statistical methods are based on the historical time data series [12]. These are simpler than physical methods but they are often limited by assumptions of normality, linearity or variable dependence [10]. Last, machine learning methods can learn nonlinear relationships between input and output data without being explicitly programmed [12].

The objective of this thesis is to study the viability of machine learning algorithms to forecast the next hour and hourly day-ahead solar irradiance. The machine learning algorithms studied are Random Forest, Feedforward Neural Network and Long Short-Term Memory networks. The study uses historical weather data from the electrical department of Tampere University located in Tampere, Finland.

2 THEORETICAL BACKGROUND

2.1 Solar Irradiance

Solar irradiance is the power per unit area received from the Sun in the form of electromagnetic radiation. The SI unit of solar irradiance is watt per square metre W/m^2 .

The study and measurement of solar irradiance is interesting for the prediction of energy generation of solar power plants. There are several measured types of solar irradiance, which are interesting for the problem discussed in this thesis:

- **Direct Normal Irradiance (DNI)**, also known as beam irradiance, is the solar radiation measured at a surface of the earth perpendicular to the sun. It only measures the direct radiation from the sun disk and excludes the diffuse radiation [11].
- **Diffuse Horizontal Irradiance (DHI)** is the radiation measured on a horizontal surface on Earth, coming from light scattered by the atmosphere. It measures radiation from all points in the sky excluding radiation from the sun disk. In the absence of atmosphere, there should be almost no diffuse sky radiation [11].
- **Reflected Radiation** is the radiation reflected by non-atmospherical elements such as the ground. However, solar panels tend to be tilted away from the reflected radiation trajectory so it rarely has relevance in the total radiation received by their surface. An exception is in conditions where the surface is surrounded by snow, which can increase significantly the reflected radiation received.
- **Global Horizontal Irradiance (GHI)** is the total irradiance from the sun on a horizontal surface on Earth. It is the sum of DHI, DNI (after accounting for the solar zenith angle of the sun θ_z) and reflected radiation. However, because reflected radiation is usually insignificant compared to direct and diffuse radiation for all practical purposes global horizontal radiation is said to be the sum of direct and diffuse radiation only [11]:

$$G = G_d + G_b \cos(\theta_z)$$

where G denotes the Global Horizontal Irradiance, G_d the Diffuse Horizontal Irradiance, G_b the Direct Normal Irradiance or beam irradiance and θ_z the zenith angle.

The mentioned types of solar radiation are shown in Figure 2.1.

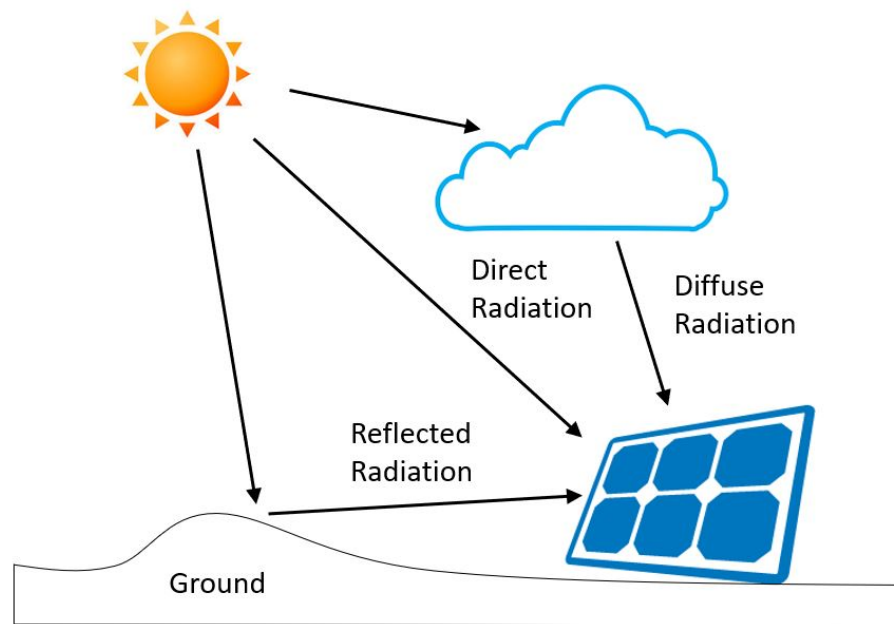


Figure 2.1. Types of solar radiation.

2.2 Machine Learning

2.2.1 Supervised Learning

Machine learning is a subfield of computer science that explores the study and construction of algorithms that can learn from data and make predictions or decisions.

Thus, instead of programming explicitly a model to solve an specific problem, the model can learn from the data and build itself. This learning process can be supervised or unsupervised.

In supervised learning some inputs and the desired output values are given to a machine learning algorithm that can learn a function that approximates the relationship between the input and the output values. Therefore, it would be like a process where the algorithm is given an exam with some questions and its correct answers so the model can learn and adjust its parameters.

On the other hand, unsupervised learning process does not have known outputs or correct answers. Instead, the algorithm has to discover itself the interesting structure in the data. Therefore, the goal of unsupervised learning is to learn the inherent structure or distribution in the data to learn more about the data.

The algorithms built and presented in this thesis are all from the supervised learning group. This is because there is GHI data available that can be used as known output. Also, because the objective of these algorithms is to model a function that can predict an output from some inputs.

In order to build a machine learning model under supervised learning, data needs to be split first in two sets: the train set and the validation set.

The model learns only from data in the train set and then, the validation set is used to evaluate its performance. The idea is that, while it is easy to overfit the train set, the performance on the validation set should match the performance obtained on new data, as long as the train and the validation set are independent.

Overfitting means that a model has learnt to predict data on the train set but gets poor performance on new data. That is because the model has learnt also the noise in the training data. In other words, it has memorised the predictions on the train set but it is unable to generalise on new data.

2.2.2 Random Forests

Before defining Random Forests it is necessary to introduce Decision Trees. The reason is that Random Forests are composed of multiple Decision Trees.

Decision Trees are flowchart-like structures that use a set of conditional rules to calculate a target value. Therefore, the decision tree asks a series of True or False questions about the data until it is confident enough to make a final prediction. However, before explaining how this flowchart of questions is defined, some basic terminology need to be introduced [6]:

1. **Root node:** it represents the entire population or sample.
2. **Splitting:** it is the process of dividing a node into two or more sub-nodes.
3. **Decision node:** it is a sub-node that splits into further sub-nodes.
4. **Leaf or terminal node:** it is a node that does not split further.
5. **Pruning:** it is the process of removing sub-nodes of a decision node.
6. **Branch or sub tree:** it is a subsection of the entire tree.
7. **Parent and child node:** when a node splits into sub-nodes, it is called parent node and the resulting sub-nodes are called child of the parent node.

The structure of the Decision Tree and some of the defined terms are represented in Figure 2.2. As mentioned above, Decision Trees create a flowchart of True or False questions. After each question, nodes are divided in two or more splits, creating different paths from the root to the leaves, where the final prediction is made.

As a supervised machine learning algorithm, Decision Trees learn from the data during the training phase and calculate the best questions to ask in order to achieve accurate predictions. In order to do that, Decision Trees try to split the nodes so that the two resulting groups are as different from each other as possible. However, the splitting process criteria is different for classification and regression trees. The Decision Tree Regressor, which is the interesting model for the objective of this thesis, normally uses Mean

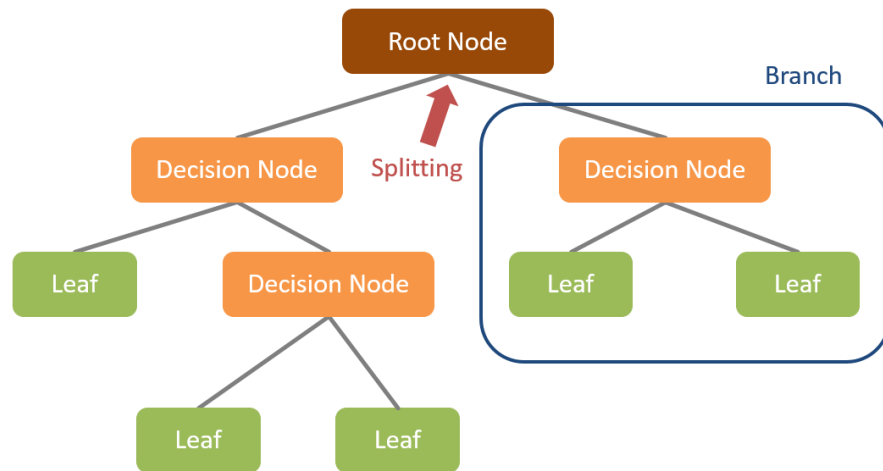


Figure 2.2. Decision Tree Structure.

Squared Error (MSE) to decide the splits. Thus, in order to find the best split, Decision Tree Regressors try every variable and every possible value of that variable and choose the split that minimizes the weighted average of the MSE of the two new nodes [6].

An invented example of Decision Tree Regressor is shown in Figure 2.3 for explanatory purposes. The model, which is meant to predict the rent price of a house, has three input variables: number of rooms, number of bathrooms and area. As it is shown, the Decision Tree splits nodes for different values of the input variables until it reaches a final prediction for the rent price.

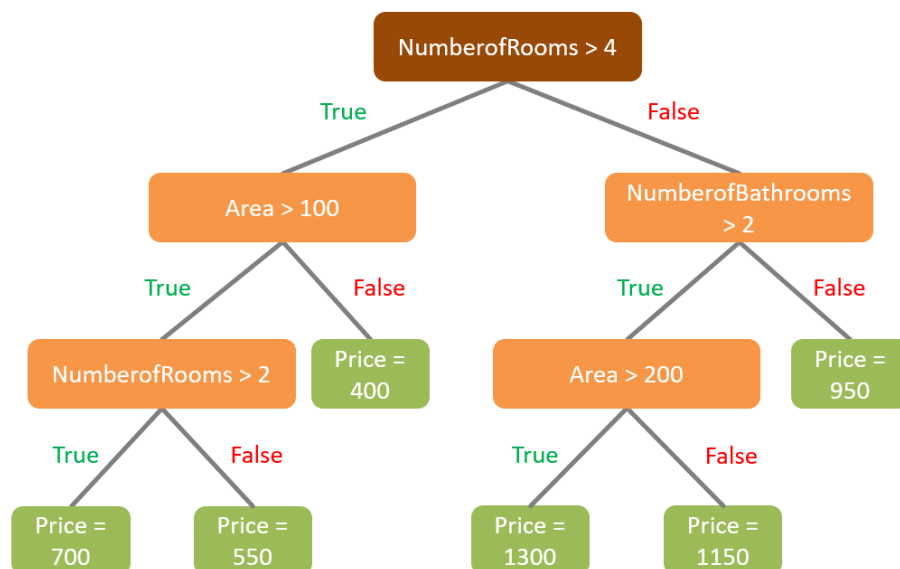


Figure 2.3. Example of Decision Tree Regressor.

The splitting process is repeated until a previously defined limit is reached or until all the leaves have just one sample in them. The last condition means that the Decision Tree has one leaf for every single observation, so no further splits are possible. Thus, MSE for the training set would be zero but the model would overfit on new data, which is not

desired.

Overfitting is a common issue on flexible models as Decision Trees. A flexible model is one whose predictions vary considerably depending on the training data. Thus, a flexible model is said to have high variance because of its sensitivity to different sets of training data. An example of a model with high variance is shown in Figure 2.4. It can be seen how the model fits even the noise in data.

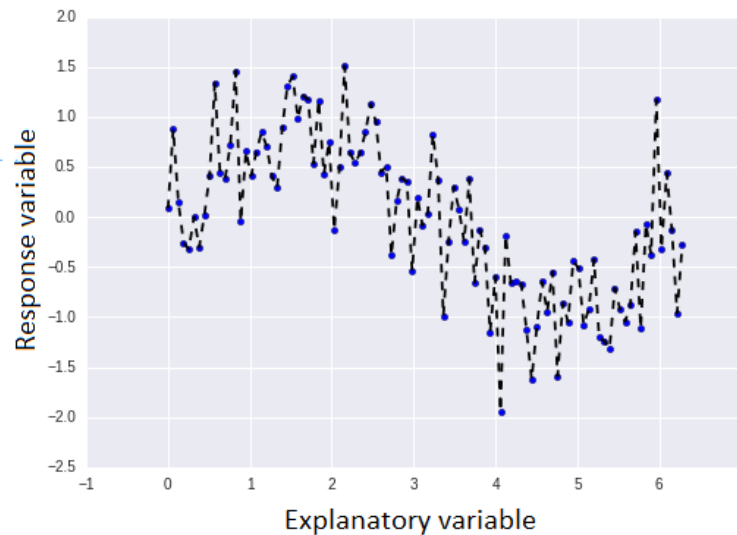


Figure 2.4. Example of flexible model with high variance and low bias.

On the other hand, an inflexible model is one that may be too rigid to fit even the training data. Therefore, an inflexible model is said to have high bias, which lead to inaccurate predictions. An example of high bias model would be a linear regression unable to fit a dataset that has non-linear pattern as shown in Figure 2.5. This is issue is called underfitting.

Summarising:

- Flexible algorithms tend to have high variance and low bias.
- Flexible algorithms train models that are accurate on average, but inconsistent.
- Inflexible algorithms tend to have high bias and low variance.
- Inflexible algorithms train models that are consistent, but inaccurate on average.

There are 3 types of prediction errors: bias, variance and irreducible error also known as noise. While the irreducible error cannot be reduced by choosing a different algorithm, bias and variance can. However, increasing the complexity of the algorithm might increase the variance while decreasing the complexity might increase the bias. This balance between bias and variance is what is known as bias-variance tradeoff. Thus, a good predictive model is one that has a good balance between bias and variance that minimizes the total error as the example shown in Figure 2.6.

As mentioned above, Decision Trees are algorithms with high variance and low bias,

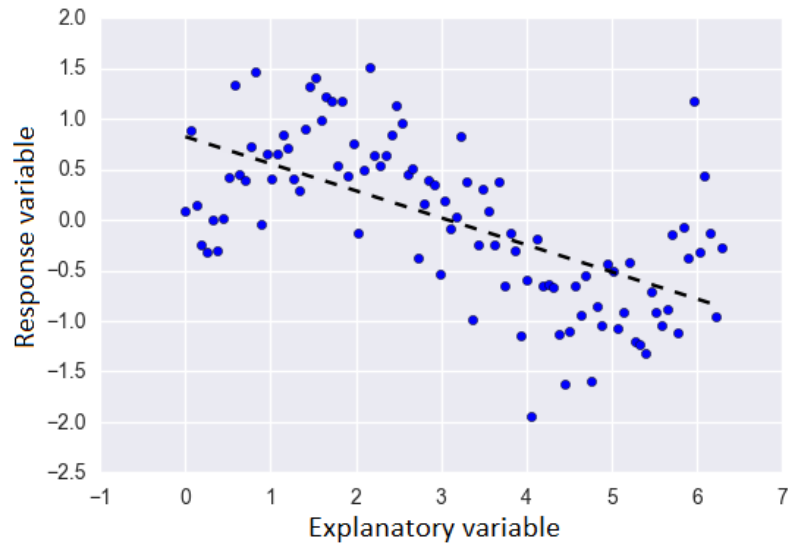


Figure 2.5. Example of inflexible model with high bias and low variance.

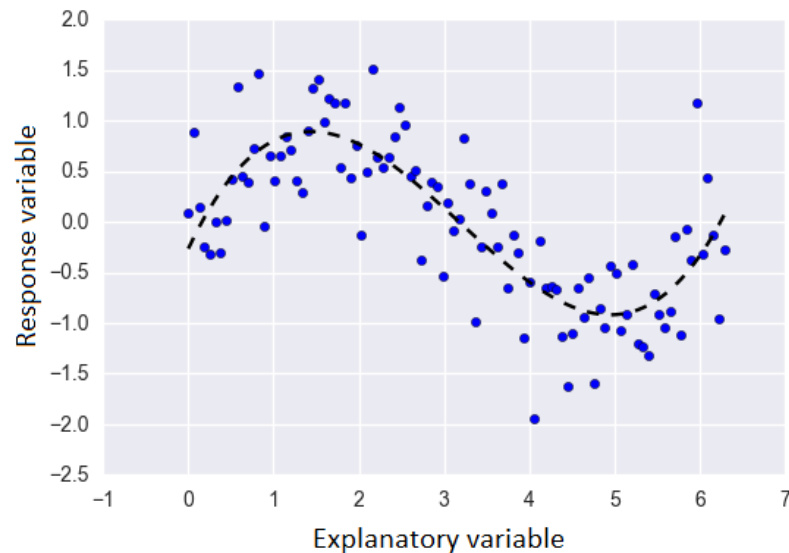


Figure 2.6. Example of model with a good balance between bias and variance.

which are prone to overfit as they can grow until they have one leaf node for every single observation. One method to reduce the variance of a Decision Tree is to limit its depth to a maximum number of splits so it cannot fit perfectly the training data. However, the variance is reduced at the cost of increasing the bias.

Unlike Decision Trees, Random Forests are not prone to overfit. Random Forest is an easy to implement machine learning algorithm, which usually produces great results on both regression and classification tasks without spending much time on hyperparameter tuning [13].

Random Forests are composed of multiple Decision Trees. This combination is made using an ensemble technique called bagging. An ensemble technique combines the

predictions from multiple models to get more accurate and stable predictions than the individual models. The simplest method to build an ensemble algorithm is to average the predictions of multiple algorithms trained on the same data. However, Random Forest does not only average the predictions of its Decision Trees. It also has two singularities that give it the name random [13]:

The first one is the ensemble technique called *bagging*. Instead of training each tree on the same data, each tree is trained on a random set of training observations. This set is the same size of the original but some samples are replaced by others of the training set. That means that there are some samples that the tree never sees while other samples are repeated. This technique of drawing samples with replacement is called *bootstrapping*. Therefore, bagging consists on training multiple models on different bootstrap samples and averaging their predictions.

The second concept is using random subsets of features for splitting nodes. Thus, instead of training each tree using all the features, they are trained using only a random subset.

Therefore, each tree has only access to a random subset of observations and a random subset of features, which adds diversity to the forest as the predictions will vary considerably for different trees. The idea of combining multiple Decision Trees trained on random observations and features is to reduce the overall variance at the cost of a small increase in the bias. Although each tree has high variance on a subset of data, their errors are not correlated due to the randomization of the training observations and features. Therefore, the errors of each tree are random and the average of a set of random errors is zero. Thus, when the predictions are averaged, the error is averaged to zero and what is left is the true relationship.

2.2.3 Artificial Neural Networks

An Artificial Neural Network (ANN) is a computational model inspired by the way biological nervous systems, such as the brain, process information. These networks are composed by neural units called neurons, which are interconnected between them. Figure 2.7 shows the structure of an artificial neuron.

This neuron is a computational unit that takes some inputs and an intercept term whose value is usually 1. Each input and the intercept term are multiplied by a weight and added later [17]. The weight for the intercept term is called bias. Then, a transfer function, also called activation function, is applied to the result. Without this transfer function, this neuron would be only a linear function. Therefore, the transfer function gives neural networks the capacity to solve non-linear problems. Some common transfer functions are the sigmoid, the hyperbolic tangent and the Rectified Linear Unit (ReLU) shown in Figure 2.8.

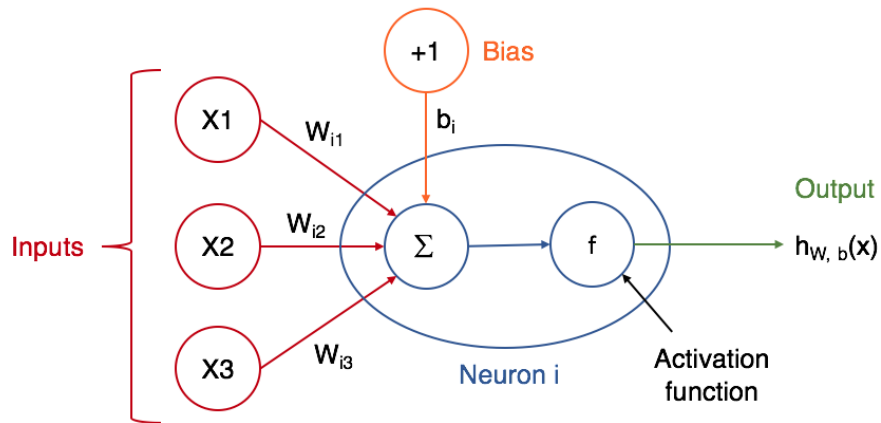


Figure 2.7. Neuron structure.

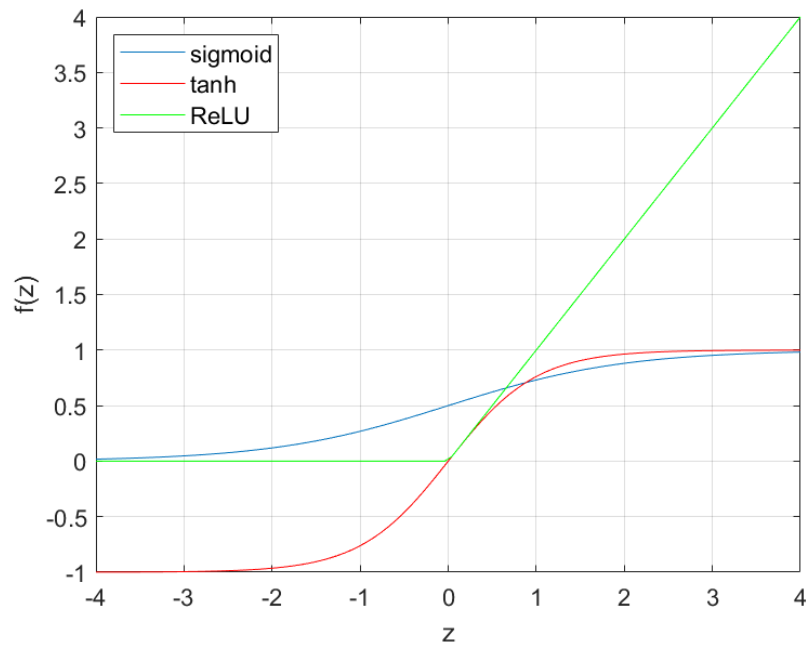


Figure 2.8. Common activation functions.

Thus, the formula [17] for the output of this neuron would be:

$$h_{W,b}(\mathbf{X}) = f_a(W_{i1}x_1 + W_{i2}x_2 + W_{i3}x_3 + b_i)$$

where $h_{W,b}$ is the output of the neuron, \mathbf{X} the input vector, x_j the input j , f_a the activation function, W_{ij} the weight j of neuron i and b_i the bias term of neuron i .

A neuron is the simplest possible neural network but more complex models can be built by interconnecting many neurons as in Figure 2.9.

As it is shown in Figure 2.9, neurons in neural networks are distributed in three types of layers:

- **Input layer:** is the leftmost layer. This is the only layer whose elements are not

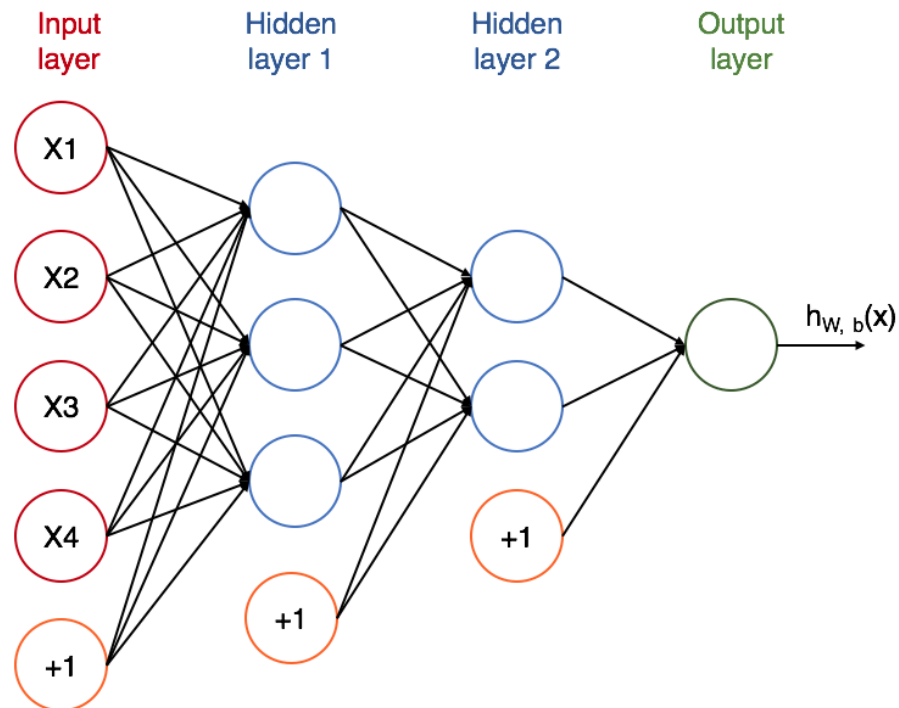


Figure 2.9. Artificial Neural Network structure.

neurons as the one described before. In this layer, each element represents an input. Thus, for the example in the figure, there are four inputs in the input layer.

- **Hidden layers:** the middle layers are called hidden layers because their output values are not observed. There are three neurons for the first hidden layer and two neurons for the second hidden layer in this example.
- **Output layer:** is the rightmost layer and its number of neurons corresponds to the number of output parameters.

The intercept term is also represented in the figure which is connected to each neuron of each layer except the input layer.

The output of each neuron is connected to all the neurons of the next layer as an input. This output is multiplied by a weight before reaching the next neuron and this weight is different for each connection.

This is an example of a fully connected Feedforward Neural Network (FNN). In FNNs the information flows only in one direction, from inputs to outputs and do not form any cycle. Fully connected means that each neuron is connected to all the neurons in the next layer. More complex neural networks can be built adding backpropagation or deleting some connections between neurons.

Neural networks are useful to solve complex and non-linear problems. However, it is difficult to know which are the main parameters that affect the response. Neural networks are like a box that transform some inputs in some outputs with the drawback that it is hard to observe or understand the inside of the box.

2.2.4 Long Short-Term Memory Networks

Long Short-Term Memory (LSTM) networks are a type of ANN. More specifically, they are a type of Recurrent Neural Network (RNN). RNNs are networks with loops in them, that make them suitable to solve problems with sequential data as time-series forecasting, speech recognition, translation, etc [16].

Figure 2.10 shows the rolled and unrolled structure of a RNN. As it can be seen, RNNs perform the same task for every element of the sequence using also information from the previous computations.

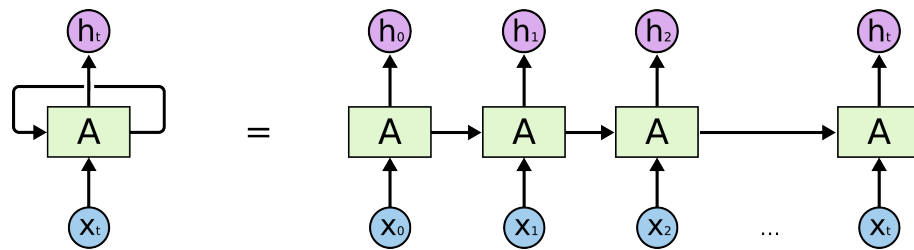


Figure 2.10. Rolled and unrolled Recurrent Neural Network [16].

Thus, RNNs are able to store memory from previous steps of the sequence so the current output is dependent on the previous calculations. However, RNNs are only able to store information from recent time-steps. In theory, RNNs are capable of handling long-term dependencies, but in practice they do not seem to learn them due to vanishing gradient problem [4]. LSTMs were explicitly designed to overcome this long-term dependency problem. They introduce new gates which allow a better control of the network memory and enable better preservation of information for long periods of time.

Like RNNs, LSTMs have a chain like structure of repeating blocks or cells. However, the main difference is inside these cells, which have 3 gates each instead of one as shown in Figure 2.11. The operation of these gates will be explained later. First, the notation used in the LSTM diagram needs to be introduced. Each line carries an entire vector. Yellow boxes represent the learned neural network layers, while pink cercles denote pointwise operations. Lines merging mean vectors being concatenated and a line forking denote its content being copied and sent to different locations.

The horizontal line on top of the diagram is called the cell state and carries information of previous intervals. Thus, it is responsible of the long-term memory. The LSTM can add or remove information of the cell state using structures called gates. These gates are composed of a sigmoid neural network layer and a pointwise multiplication operation to decide what information is let through. LSTMs have 3 gates:

1. **Forget gate:** it decides what information is removed from the cell state. The output is a vector of values between 0 and 1 telling how much each cell state value should be forgotten. If a value of the cell state is multiplied by 0 that value is completely forgotten, while if it is multiplied by 1 the value is completely kept. The output of the

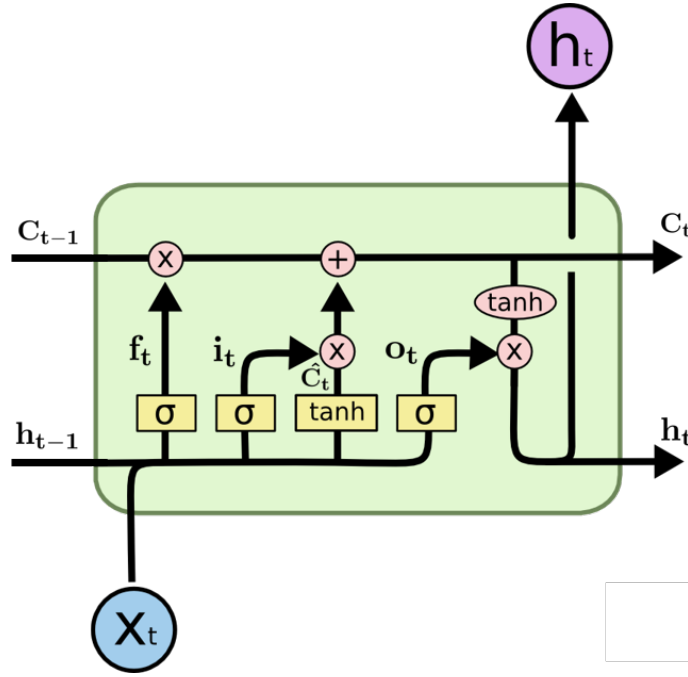


Figure 2.11. LSTM cell diagram [16].

forget gate is calculated as follows:

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$$

where f_t is the forget gate output vector, σ the sigmoid function, x_t the input vector at time t , h_{t-1} the previous hidden state, W_f are the weights of the forget gate, and b_f the biases of the forget gate.

2. **Input gate:** it decides what new information is stored in the cell state. This gate is composed of two parts. First is the input gate layer composed by a sigmoid layer. It outputs values between 0 and 1 in order to decide how much a value is updated. The second is the input modulation gate layer composed by a tanh layer. This layer creates a vector of new candidate values that could be added to the cell state. The outputs of both layers are calculated as follows:

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$

$$\hat{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$$

where i_t is the input gate output, W_i are the weights of the input gate, b_i the biases of the input gate, \hat{C}_t the new cell state candidates, W_C the weights of the input modulation gate and b_C the biases of the input modulation gate. Thus, in order to update the old cell state C_{t-1} to the new cell state C_t , the LSTM first forgets the cell state values decided by the forget gate. Then, the new candidate values scaled by

the input gate output are added to the cell state. Thus, the equation for this update process is:

$$C_t = f_t \odot C_{t-1} + i_t \odot \hat{C}_t$$

3. **Output gate:** it decides what parts of the cell state are output using a sigmoid layer. The output of the sigmoid layer is calculated as:

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o)$$

where o_t is the output of the output gate, W_o are the weights of the output gate and b_o the biases of the output gate.

The cell state is passed first through a \tanh layer to push the values to be between -1 and 1. Then, it is multiplied by the output of the output gate, so that only the selected parts of the cell state are output. Therefore, the output h_t of the cell called hidden state is calculated as follows:

$$h_t = o_t \odot \tanh(C_t)$$

The whole process is repeated for each element of the sequence. Weights and biases are updated by the model by minimizing the error score between the LSTM outputs and the actual training observations.

2.3 Time Series Data

2.3.1 Baseline Predictions

When building a forecasting model, it is useful to have a baseline model to compare how well the new model performs and quantify the prediction improvement if obtained.

A common baseline model in solar irradiance forecasting is the persistence algorithm. The persistence algorithm simply sets the value at time t in the previous day $d-1$ to be the prediction value at the same time t in the day d [12]. Thus, this algorithm is free of training and parameters setting.

2.3.2 Multiple Train and Validation Splits

Typically, data is split into train and validation sets randomly. However, there are some problems when applying this method to time-series data.

First is that in real world applications, previous observations are usually used to predict future observations. A random split does not preserve the time ordering, which leads the model to make some predictions having trained on posterior observations.

However, the most important problem is that time-series data is often strongly correlated along the time axis. That means that each observation is not independent from each other and we cannot simply randomly split the data. Therefore, the temporal order must be respected. Thus, the first part of the data is used for training and the last for validation.

Often, many different splits between training and validation set are made in order to get a better estimation of the performance on new data. This method is also often used to optimize hyperparameter values of the model. However, if we want to respect the temporal order and for a fixed validation set size, there is only a single possible train and validation split.

This problem is solved with the next algorithm. First, data is split into k equal blocks and the validation set size is fixed (e.g., one block). Then, several folds with different train and validation splits are defined. The first fold starts with a small subset of p contiguous blocks of data used for training and another subset containing the next values is used to evaluate the performance. Note that data after validation set is ignored. The second fold splits data into $p+1$ blocks for training and the next block for testing. This process is repeated until all the k blocks of data are used in the last fold. Thus, the same observations used for validation in a fold are included for training in the next fold. An example of this algorithm is shown in figure 2.12 for a number of 5 folds. Note that the size of the train set increases in every fold while the validation set size stays constant.

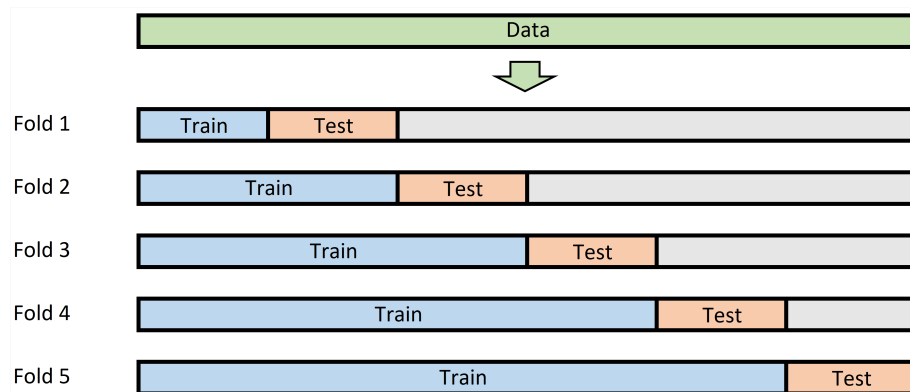


Figure 2.12. Example of multiple train and test splits for time-series data.

Therefore, a model is trained and evaluated for each fold and the average performance is calculated. Thus, this algorithm requires creating multiple models, which improves the performance estimation at additional computational expense.

2.4 Hyperparameter Optimization

Sometimes it can be difficult to choose the right hyperparameters for a machine learning algorithm. While parameters are internal to the machine learning algorithm and learnt automatically (e.g. neuron weights of a neural network), hyperparameters are set by the operator of the neural network and have a huge impact on the model performance (e.g. number of neurons).

The optimization of these hyperparameters is a process that can be very time consuming because the list of hyperparameters can be large. Moreover, finding the optimal values is non-trivial and usually the optimization process is stopped when a solution close to the optimal is achieved.

There are several methods of hyperparameter optimization:

- **Manual tuning:** the simplest way to select hyperparameters of a machine learning model is manually by trial and error. It is a method commonly used and effective with skilled operators. However, it is not scientific and it is difficult to know if the hyperparameters are fully optimized.
- **Grid search:** this algorithm involves defining first a set of hyperparameter values. Then, a model is trained and tested for all possible hyperparameter combinations and the best one is selected. While this algorithm is very simple to implement, it is a good choice only when the time required to train the model is low. Moreover, the number of trials grows exponentially with the number of hyperparameters. For example, if it is desired to make 10 evaluations in order to optimize 4 hyperparameters, then, the algorithm will make 10^4 evaluations. Instead, if it is desired to optimize 5 hyperparameters, 10^5 evaluations would be performed.
- **Random search:** the idea is similar to grid search, but instead of trying all possible combinations only a defined number of combinations randomly selected are evaluated. As grid search, random search is easy to implement but provides higher accuracy with less training cycles for problems with high dimensionality [3]. However, if not enough number of trials are run, the hyperparameter space may not be fully covered. Moreover, this algorithm may still not be suitable for long training algorithms such as large neural networks.

There are more complex optimization techniques that choose the next combination based on the results obtained in previous tests. Thus, instead of trying random combinations, these techniques try the combinations that seem more promising. However, for simplicity, only the mentioned optimization techniques were used in this thesis.

2.5 Performance Metrics

Commonly used metrics to evaluate forecast accuracy are the Mean Absolute Error (MAE) and the Root Mean Squared Error (RMSE). MAE measures the average error and is calculated as the mean value of the sum of absolute differences between actual and predicted observation:

$$MAE = \frac{1}{n} \sum_{i=1}^n |y_i - \hat{y}_i|$$

where y_i is the actual, \hat{y}_i is the predicted and n is the number of observations. On the other hand, RMSE is the square root of the average of squared differences between actual and predicted observation:

$$RMSE = \sqrt{\frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2}$$

Both metrics have the advantage that they use the same units as the predicted feature. From an interpretation point of view, MAE is easier to understand. However, RMSE gives high weight to large errors since the errors are squared before being averaged. Therefore, RMSE can be useful when large errors are not desired.

2.6 Used Software

There are different programming languages to create machine learning models. Currently, Python is the most used in machine learning projects and has several libraries dedicated to machine learning.

One of them is the scikit-learn library, which has several tools for data mining and data analysis. This library was used to create Random Forest models.

On the other hand, the Keras framework was chosen to create neural network models. Keras is an open source neural network library for Python and nowadays it is the most popular due to its simplicity. Therefore, it has a large community of users. Moreover, Keras can run on top of the second most popular machine learning framework Tensorflow from Google.

Matlab software was used to import data from the database and also to perform most of the data preprocessing.

3 RESEARCH METHODOLOGY

3.1 Weather Data and Sensors

In order to build a machine learning model to forecast solar irradiance, it is needed to gather a sufficient amount of weather data.

The department of electrical engineering of Tampere University has a solar PV power station research plant at the top of the department building which consists of 69 PV modules. This research plant is provided with climate measuring system which collects weather, irradiance and PV module temperature measurements continuously with a 10 Hz sampling frequency [15].

The weather station includes measurements of ambient temperature, relative humidity, wind speed and direction, as well as global and diffuse solar irradiances on the horizontal plane.

Solar irradiance is measured with the pyranometer CMP22 (Kipp&Zonen) and diffuse irradiance with the pyranometer CMP21 (Kipp&Zonen) combined with a shadow ring CMC121 (Kipp&Zonen), blocking the direct solar radiation. Wind speed and direction are measured with an ultrasonic wind sensor WS425 (Vaisala), and ambient temperature and humidity are measured with the sensor HMP155 (Vaisala). All these sensors are shown in figure 3.1.

All the climatic data is acquired, transmitted and recorded continuously in a database for future analysis. The data is accessed using SQL queries from any computer connected to the local area network of the department.

As mentioned above, data is collected with a 10 Hz sampling frequency. However, importing data with such an elevated resolution might be unnecessary for this problem and inconvenient due to the long time it would take to import all the data and the disk memory required.

In fact, there is a maximum amount of data allowed to be imported at once by the system. Due to this limitation, a compromise between sampling frequency and required resources had to be set. The higher is the sampling frequency the more information is available but also the more required resources. Therefore, the sampling frequency chosen was 1 minute which was considered high enough as the irradiance values tried to predict have hourly frequency.



(a) Pyranometer CMP22



(b) Pyranometer CMP21



(c) Sensor HMP155



(d) Ultrasonic wind sensor WS425

Figure 3.1. Weather station sensors.

Thus, all data available since June 2011 of GHI, DHI, ambient temperature, humidity, wind speed and wind direction was imported with a sampling frequency of 1 minute. The software used to import the data was Matlab which was also used for some preprocessing steps that will be mentioned later.

3.2 Data Preprocessing

Usually, data gathered in real world is not perfect and needs to be preprocessed before being used. Data preprocessing involves cleaning data, adding features, filling missing values and other techniques that will be explained.

First step is to check that sensor readings are correct. Some misreadings can be found in the dataset as there are negative GHI, DHI or humidity readings. A simple method to clean the data is to remove the misreadings and replace them with an interpolation of preceding and succeeding points [2].

Weather parameters have both daily and yearly seasonality. Thus, it could be interesting

to add variables, which the model can use to interpret day and year values.

The most simple approach would be to create several variables: day of the year, hour, minute, etc. However this has the drawback that the variables are not continuous. For example, if the year changes, the variable day of the year jumps from 365 to 1. There is just one day of difference between day 365 and day 1, but for the machine learning model could appear to be 365 days of difference. For that reason, a different approach was set.

Two variables were created for each day of the year and day hour to transform them into 2 dimensions using the sine and cosine functions. The transformation for time of the day is calculated as follows:

$$t_{cos}(t) = \cos\left(\frac{2\pi t}{N_s}\right)$$

$$t_{sin}(t) = \sin\left(\frac{2\pi t}{N_s}\right)$$

where t denotes the time starting from midnight and N_{tf} the number of time fractions in a day in the same units as time

This transformation for time of the day would look like the "clock" in Figure 3.2.

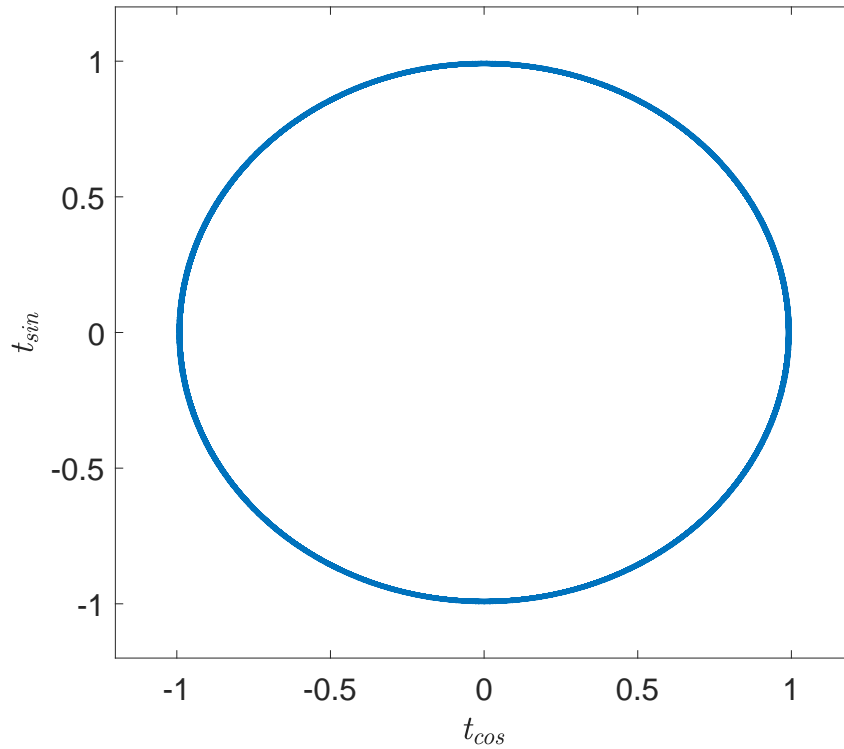


Figure 3.2. Time of the day transformation into features t_{cos} and t_{sin} .

And the transformation for the day of the year is:

$$d_{cos}(d) = \cos\left(\frac{2\pi d}{N_d}\right)$$

$$d_{sin}(d) = \sin\left(\frac{2\pi d}{N_d}\right)$$

where d denotes the day of the year and N_d the number of days in that year, whose values are 365 or 366. This transformation would also look similar to the one in Figure 3.2 but representing dates from the beginning to the end of the year.

Also, after exploring the dataset it was found that there were some missing values in the time-series data. That is that some readings were lacking and there was a timestamp greater than 1 minute between some samples.

As said before, time-series data should be continuous and have the same time distance from sample to sample. Thus, it was decided to fill the missing values in data for small gaps and cut where the gaps were too large to make a good approach.

The first step was to add the missing timestamps to the dataset with Not a Number (NaN) values for every measurement during those timestamps. This is shown in figure 3.3 for GHI measurements.

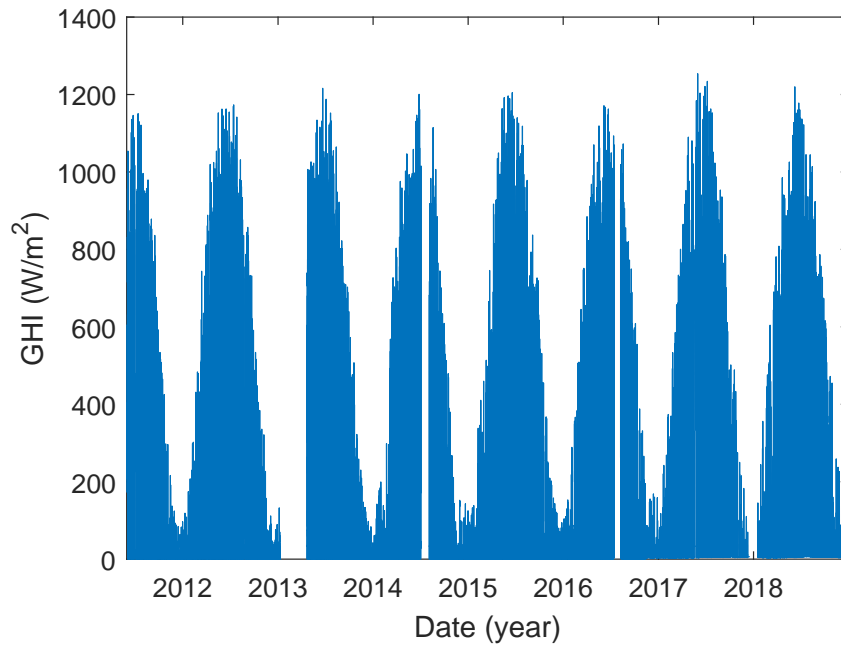


Figure 3.3. Missing values in dataset.

Next step was to locate every gap with NaN values in the dataset and find its size in minutes. In total, 2 062 gap regions were found. Most of these gaps had a size of a

few minutes and some had a size of several hours or days. The last were due to some maintenance in the weather station.

Thus, the smaller gaps were filled using a moving window mean with a length of 20 minutes. After that only 123 gaps were left.

Then, it was considered that days with gaps smaller than 12 hours and with no missing values during midday, could be fairly well estimated. Therefore, those gaps with size smaller than 12 hours were checked manually. It was found that many readings were lacking during night so values for GHI and DHI were easily filled as zero during night time. Then, if the rest of the gap samples were not missing during midday, they were estimated using a shape-preserving piecewise cubic spline interpolation.

However, some gaps were too large to be estimated without making a considerable error. For that reason the data was split where gaps larger than 12 hours were found. Moreover, the gaps were often close to each other. When this happened, all the days from the first gap to the last were removed from the dataset.

Nonetheless, before cutting the data, this was downsampled from one minute to one hour for several reasons. First is that every imported minute sample consisted of a single random sample. Thus, this value could be far from the real average value for the minute measured. So averaging the 60 minute samples to get the average hourly measurements would reduce noise in data.

Another reason is that climate variables are expected to exhibit a 24 hours seasonality. Therefore, a vector for every feature of at least the 24 hour previous measurements seems a good choice to forecast the next GHI values. However, a sampling frequency of 1 minute would require a vector of size 1 440 for every feature, which is computationally demanding for training the machine learning algorithms [5]. Thus, downsampling to hourly averaged measurements would reduce vector size from 1 440 to 24 samples.

The last reason is that as the values to be predicted are hourly averaged, it might be easier for the machine learning algorithms to learn also from hourly averaged values.

Once the data was downsampled to one hour and the missing values filled, there were still some misreadings in the DHI measurements. Sometimes the measurements were too high which probably meant that the shadow ring was not configured correctly so the sensor was receiving direct irradiance as shown in figure 3.4.

Most of these misreadings were located in order to be removed later. Then, all the large gaps and DHI misreadings were removed so that 81,51% of the original amount of data was left. The final amount of data was equivalent to 2 259 days or 54 216 hourly averaged measurements.

Removing non-useful data added some discontinuity to the time series data. However, this is not a problem as long as there are no time jumps inside samples. As said above, every data sample will be composed of limited length sequences containing the last 24

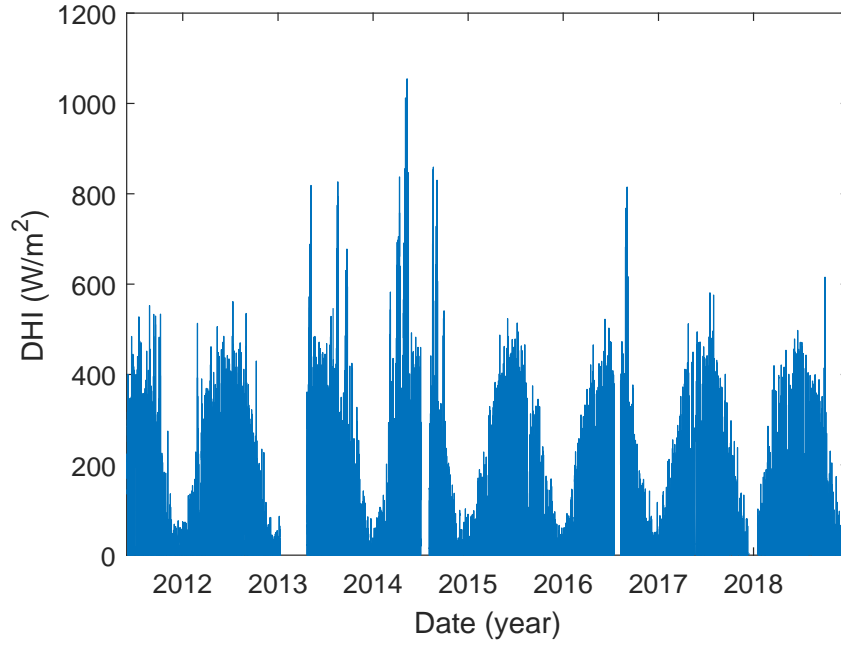


Figure 3.4. Diffuse Horizontal Irradiance. Abnormal high values due to missreadings in CMP21 sensor.

observations. Therefore, data was carefully removed so that there was no discontinuity inside samples.

Finally, the last step of the data preprocessing is to normalize all the data so that all the values are in the range within 0 and 1. Input variables are measured in different units, which means the variables have different scales. That difference in the scale across the input variables might affect the training of the algorithm. For example, large input values may result in a model that learns large weight values, which often is unstable and results in higher generalization error.

A value is normalised as follows:

$$z_{norm} = \frac{z - z_{min}}{z_{max} - z_{min}}$$

Where z_{max} and z_{min} are the maximum and minimum values of the feature containing the z value.

All the preprocessing steps are summarized in figure 3.5.

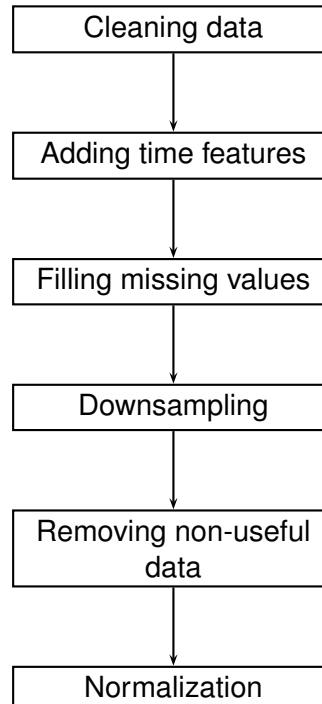


Figure 3.5. *Data preprocessing steps.*

3.3 Train and Validation Data

All data available will be used for training except the two last years, which will be used for validation. Thus, considering the data eliminated, the number of samples in the train and validation sets are 38 856 and 15 360 respectively. Therefore, 71.7% of the observations were used for training and 28.3% for validation.

The output data dimensions will be different for each forecast. Output data for next hour forecast will only have one feature with the target GHI next hour, while output data for day ahead forecast will have 24 target values, one for each of the next 24 hours.

Finally, despite the mentioned splitting choice, multiple train and validation splits will be used for hyperparameter optimization. However, only the train set will be used during the optimization process. Then, once the optimal configurations are found they will be trained with the whole train set and tested on the validation set. The reason is to reduce the influence of the validation set on the hyperparameter selection. Thus, the calculated validation error should be very close to the real error. The mentioned splits are listed in Table 3.1.

As can be seen, the train set will be split into 4 equal subsets to make 3 different folds of train and validation data used during hyperparameter optimization. It is also possible to see that although it was said that the last 2 years of data would be used for validation, the real amount of validation data is 1.75 years. This is because despite the selected

Table 3.1. Train and validation splits for hyperparameter optimization and final model.

Splits	Train set [samples]	Validation set [samples]	Train data [%]	Validation data [%]	Train data [years]	Validation data [years]
Fold 0	1 to 9 714	9 715 to 19 428	17.9	17.9	1.1	1.1
Fold 1	1 to 19 428	19 429 to 29 142	35.8	17.9	2.2	1.1
Fold 2	1 to 29 142	29 143 to 38 856	53.7	17.9	3.3	1.1
Final	1 to 38 856	38 857 to 54 216	71.7	28.3	4.4	1.75

validation set starts in January 2017 and ends in December 2018, some data was withdrawn during data preprocessing. However, it is enough to calculate the prediction error accurately.

3.4 Persistence Algorithm

Once the data is preprocessed and ready to work with, the persistence algorithm explained in chapter 2 was used to build a baseline model. This baseline model will be useful to analyse the prediction improvement of the machine learning algorithms.

The prediction error of the persistence algorithm was obtained from the same test data that will be used to evaluate the different machine learning algorithms, in order to make the results comparable.

MAE and RMSE are used to represent the error of the results as they have the same units as the predicted value.

Figure 3.6 shows an example of values predicted by the persistence algorithm and its respective targets for several days of validation data.

As it can be seen, the persistence algorithm has good accuracy for consecutive sunny days, but performs poorly when there are changes in weather.

3.5 Random Forest Experiments

3.5.1 Variable selection

Random Forests are a good tool to find important variables with greater impact on the output [7]. When training machine learning algorithms, more inputs mean more training time. Therefore, it would be interesting to eliminate those input variables with less importance in the prediction. Moreover, it is interesting for research purposes to know which variables are useful to predict GHI on both forecasts: next hour and day ahead.

Data available is time-series. Thus, there are some important design choices for each

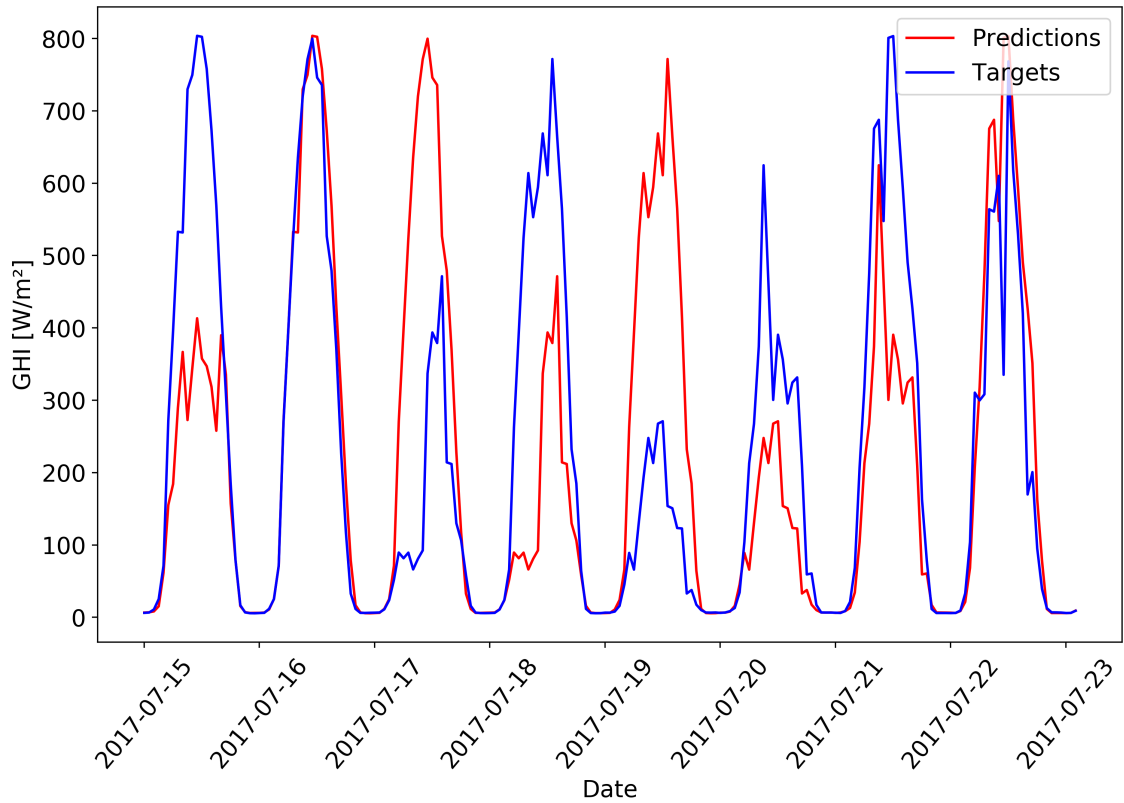


Figure 3.6. Example of persistence algorithm implementation.

feature [5]:

- use only the last measurement before the prediction as input.
- use a vector of past values as input.

The first option can lead to miss information useful for the prediction as it only considers the last value available. Therefore, using a vector with several past measurements seems a better option. However, the vector size has to be large enough to achieve a satisfactory performance, but remain as small as possible to avoid excessive training time [5].

Since irradiance exhibits daily seasonality, a vector containing measurements from the last 24 hours is chosen [5]. As the sampling frequency is one hour, the vector size is 24 measurements for each feature.

Thus, data has to be prepared so there are 4 vectors with 24 values each for variables DHI, GHI, ambient temperature and humidity as well as 2 variables to represent time of the day and other 2 for day of the year. That makes a total amount of 100 input variables. Wind speed and wind direction features were withdrawn from dataset as they usually have weak correlation with solar irradiance as shown in several papers [5, 9, 12, 18]. Data is arranged so each row is a sample and each column is a feature or variable. Moreover, the samples are overlapped in moving windows with 1 hour increments. That means that each sample is like its previous one but moved one hour to the future.

Once data is prepared a Random Forest Regressor for each forecast is created using the

Sci-kit learn library from python. Random Forest hyperparameters are set to default in this occasion as the main goal here is to find the importances of each variable instead of getting a model with high performance. However, the performance value will be saved for comparison purposes.

The importances found rounded to four decimals are shown in Table A.1 in Appendix A. Each variable is given an importance with a value from 0 to 1 and the sum of all the importances is 1. Variables with several timesteps are labeled with a specific nomenclature to identify the timestep. Time t corresponds to time at next hour. Therefore, observations one hour before predictions are labeled with $t-1$, observations 2 hours before predictions with $t-2$, and so on until observations of previous 24 hours, which are labeled with $t-24$.

As shown, the variable with highest importance (0.854) is the GHI at $t-1$, which is normal as values of GHI at times t and $t-1$ are usually very similar. The following variables are different measurements of GHI and DHI. Then, ambient temperature has a lower effect and humidity has almost no relevance in the forecast. In fact, the importance values rounded to the third decimal obtained for humidity variables were all zero.

It is also remarkable that the variable t_{cos} had also poor relevance. However, there is a possible explanation for that. The decision trees that form a Random Forest are not trained with all the variables, which means that each tree is trained with just a few of them. Thus, there is a great chance that the decision trees are trained with just one of the two variables that form each bidimensional variable that represent time. If this variables are not input together, the decision tree cannot understand them and gives them less importance.

The same experiment was run to find the important variables for the day ahead forecast. The results are shown in Table A.2 in Appendix A. Here, the last GHI observation does not have as much importance as for the next hour forecast, but it is still the variable with highest importance (0.2011). Then time, GHI and DHI variables had most of the importance in total, followed by temperature and humidity in last place again.

Therefore, due to the low importance of humidity and in order to reduce training time, humidity variables were excluded. The variables left were:

- Last 24 hours DHI (variables 1 to 24).
- Last 24 hours GHI (variables 25 to 48).
- Last 24 hours ambient temperature (variables 49 to 72).
- Time of the day (variables 73 to 74).
- Day of the year (variables 75 to 76).

3.5.2 Next Hour Forecast

The first algorithm tested was Random Forest. Random Forest is easy to implement and usually gives good performance with minimum time spent on hyperparameter tuning. For that reason, it is a good start point. Moreover, it was used to find the variable importances, so most of the code was already written. In order to find a Random Forest that predicts the next hour GHI with a decent performance, several experiments selecting different ranges of hyperparameter values were run. The set of hyperparameters adjusted were:

1. *n_estimators*: the number of trees in the forest.
2. *max_depth*: the maximum depth of the tree.
3. *min_samples_split*: the minimum number of samples required to split an internal node.
4. *min_samples_leaf*: the minimum number of samples required to be at a leaf node.
5. *max_features*: the number of features to consider when looking for the best split:
6. *bootstrap*: whether bootstrap samples are used when building trees.

Before optimizing the hyperparameters, a basic configuration of hyperparameters was tested in order to measure later the reduction in error after hyperparameter tuning. The test was run using 3 different folds of train and validation splits. Therefore, 3 models with the same default hyperparameters were trained and tested on different splits of data and the average MSE was calculated. The values set for the base Random Forest model are shown in Table 3.2.

Table 3.2. Random Forest base model hyperparameters.

Identifier	Hyperparameter	Value
1	<i>n_estimators</i>	100
2	<i>max_depth</i>	None
3	<i>min_samples_split</i>	2
4	<i>min_samples_leaf</i>	1
5	<i>max_features</i>	'auto'
6	<i>bootstrap</i>	True
mean MSE		0.002739

The MSE obtained was 0.002739. Note that this is the error obtained with the normalized variables. In order to get the real MSE, the outputs and targets have to be transformed back to their real scale and then calculate the error. However, at this moment it is not necessary as the objective is to optimize the hyperparameters. Later, the real MSE will be calculated with the optimal configuration.

Then, several configurations of hyperparameters were tested using the Random Search method. Moreover, each configuration was tested on 3 different folds of data as done with

the base model. The parameter grid to random search had the ranges of values shown in Table 3.3.

Table 3.3. First search hyperparameter values. Random Search with 300 iterations.

Identifier	Hyperparameter	Value Ranges	Best Found
1	<i>n_estimators</i>	20, 40, 60, 80, 100, 120, 140, 160, 180, 200, 500	200
2	<i>max_depth</i>	10, 20, 30, 40, 50, 60, 70, 80, 90, 100, None	50
3	<i>min_samples_split</i>	2, 5, 10, 15, 100	5
4	<i>min_samples_leaf</i>	1, 2, 5, 10	10
5	<i>max_features</i>	0.33, 'auto', 'sqrt'	'auto'
6	<i>bootstrap</i>	True, False	True
mean MSE			0.002649

In the case of *max_features*, 0.33 means that only a third of the features are considered at each split, 'auto' that all the features are considered and 'sqrt' that only the root square of the number of features are considered. While 'sqrt' usually gives good results in classification, 0.33 is recommended for regression [8]. However, using 'auto' usually gives better results empirically.

Altogether, there are $11 \cdot 11 \cdot 5 \cdot 4 \cdot 2 \cdot 2 = 9680$ possible combinations. However, Random Search can find good configurations without testing all of them. In total, 300 configurations were tested on 3 folds, so 900 iterations were run in total. The MSE of the best model found was 0.002649 which means an improvement of 3.29%.

Some interesting hyperparameters affecting considerably the performance were *bootstrap* and *max_features*. As shown in Figure 3.7, models using bootstrap samples and with access to all the features produced the results with lowest MSE.

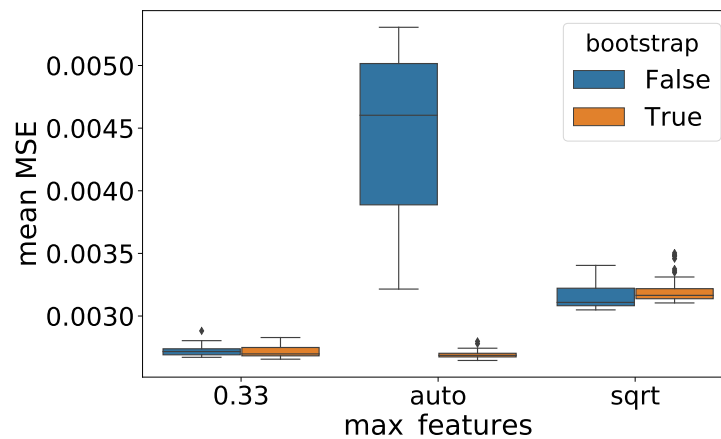


Figure 3.7. Relationship between *bootstrap*, *max_features* and mean MSE.

For that reason, it was considered to run a second Random Search fixing *bootstrap* to True and *max_features* to 'auto'. But first, the rest of the hyperparameters were studied only for the models obtained with this configuration in the first Random Search. Figure 3.8 shows the results of this exploration. As shown in Figure 3.8a, the MSE

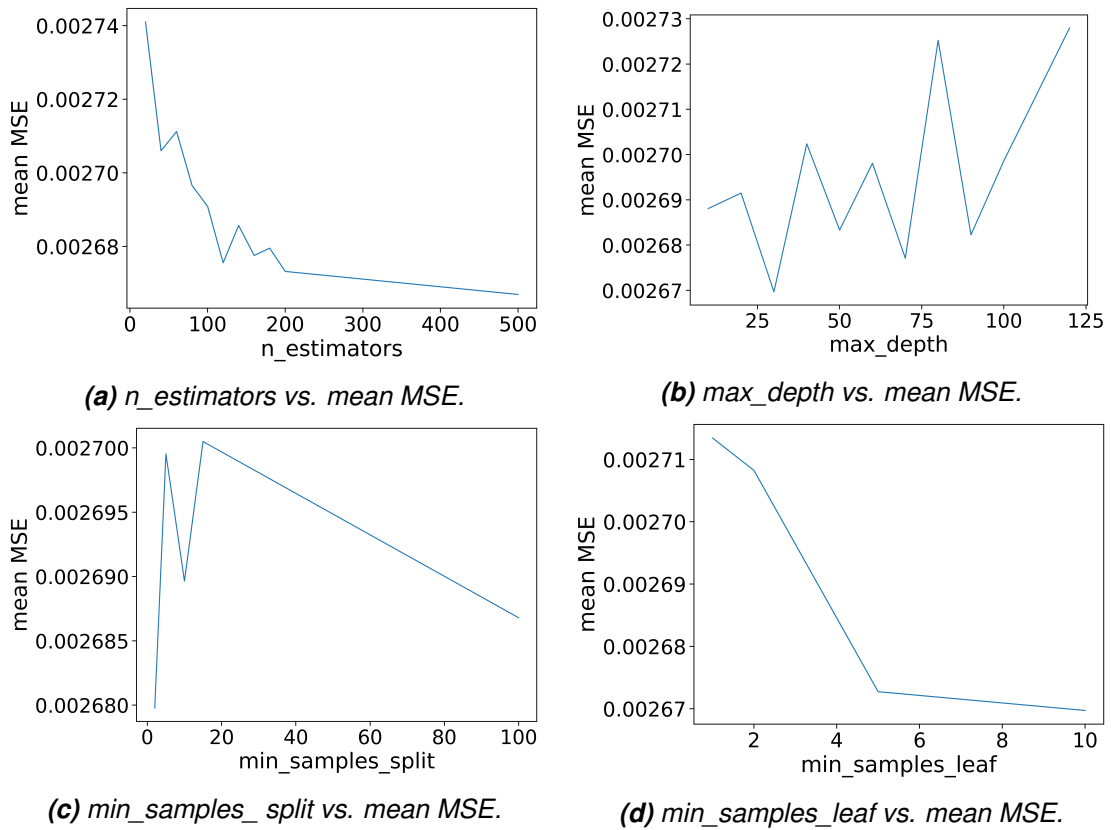


Figure 3.8. First search results.

decreases considerably for $n_estimators$ higher than 100, but after then, the improvements get smaller. Watching at Figure 3.8d it seems promising to explore values of $min_samples_leaf$ greater than 10. Instead, max_depth do not seem to affect the MSE and $min_samples_split$ is difficult to interpret.

Therefore, a second Random Search of 200 iterations was run. The ranges of values explored and the best configuration found are shown in Table 3.4.

Table 3.4. Second search hyperparameter values. Random Search with 200 iterations.

Identifier	Hyperparameter	Value ranges	Best Found
1	$n_estimators$	100, 150, 200, 250, 300, 500	300
2	max_depth	10, 20, 30, 40, 50, 60, 70, 80, 90, 100, None	10
3	$min_samples_split$	2, 5, 10, 15, 20, 30, 100	5
4	$min_samples_leaf$	5, 10, 20, 30	15
5	$max_features$	'auto'	'auto'
6	$bootstrap$	True	True
mean MSE			0.002641

The average MSE obtained by the best model found in the second Random Search was 0.002641, which is slightly better than the MSE obtained in the first Random Search. Therefore, further exploration might not be beneficial enough and the optimization process can conclude. A summary of the hyperparameter optimization process is found in

Table 3.5.

Table 3.5. *Summary of Random Forest hyperparameter optimization process for next hour forecast.*

Model	mean MSE	Improvement (%)
Base model	0.002739	—
First Random Search	0.002649	3.29
Second Random Search	0.002641	3.58
Final	0.002641	3.58

As it can be seen, the optimized model did not perform considerably better than the base model. As said before, Random Forests are known to produce good results with minimal hyperparameter optimization.

Finally, a Random Forest with the best configuration found was trained and tested on the train and validation splits. The results for this and the rest of algorithms are shown in Chapter 4.

3.5.3 Day Ahead Forecast

A similar process was followed to create a Random Forest able to forecast the day ahead GHI. However, in this problem the algorithm has to forecast 24 values at the same time. Therefore, the prediction error will be the average of the 24 predictions MSE. Thus, it is expected to be larger than the next hour forecast MSE.

First, a base model was created with the hyperparameters listed in Table 3.6 and the average MSE obtained was 0.008877.

Table 3.6. *Random Forest base model hyperparameters.*

Identifier	Hyperparameter	Value
1	<i>n_estimators</i>	100
2	<i>max_depth</i>	None
3	<i>min_samples_split</i>	2
4	<i>min_samples_leaf</i>	1
5	<i>max_features</i>	'auto'
6	<i>bootstrap</i>	True
mean MSE		0.008877

Then, a first Random Search was run with the hyperparameter values listed in Table 3.7. A total of 100 configurations were explored and the best model found produced a MSE of 0.008404. The models were analysed again depending on their *bootstrap* and *max_features* values as illustrated in Figure 3.9. It is remarkable that unlike in next

hour forecast, the models that produced best results were those with *bootstrap* True and *max_features* 0.33.

Table 3.7. First search hyperparameter values. Random Search with 100 configurations.

Identifier	Hyperparameter	Value Ranges	Best Found
1	<i>n_estimators</i>	100, 150, 200, 250, 500	150
2	<i>max_depth</i>	10, 20, 30, 40, 50, 60, 70, 80, 90, 100, None	None
3	<i>min_samples_split</i>	2, 5, 10, 15, 20, 30, 100	30
4	<i>min_samples_leaf</i>	8, 10, 15, 20, 30	20
5	<i>max_features</i>	0.33, 'auto', 'sqrt'	0.33
6	<i>bootstrap</i>	True, False	True
mean MSE			0.008404

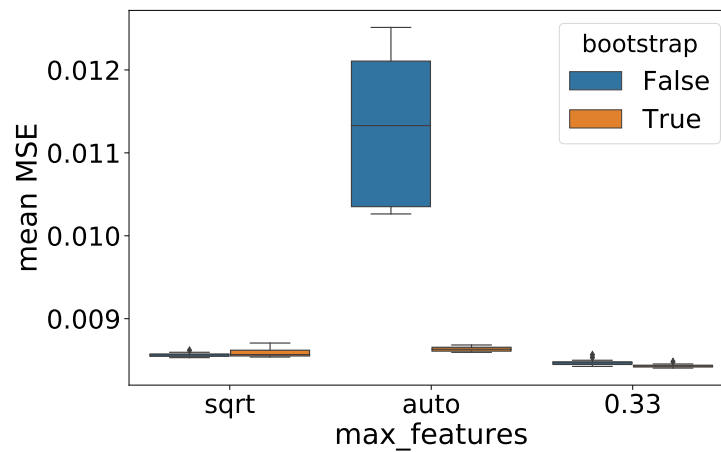


Figure 3.9. Relationship between *bootstrap*, *max_features* and mean MSE.

The rest of hyperparameters were analysed for these models in order to narrow the ranges of values explored. As it can be seen in Figure 3.10, values of *min_samples_split* close to 20 seem promising. A similar relationship can be seen for *min_samples_leaf* where values close to 20 produced the best results. No clear relationships could be found for the rest of the hyperparameters although the error is expected to decrease for higher number of *n_estimators*. This relationship cannot be observed probably because the number of samples studied is too small. However, the hyperparameters will be analysed again after the second Random Search.

Thus, a second Random Search of 100 more iterations was run. Hyperparameters *bootstrap* and *max_features* were fixed to True and 0.33 respectively. Moreover, values for *min_samples_split* and *min_samples_leaf* were narrowed to values close to 20. The hyperparameter values explored are listed in Table 3.8 and the MSE of the best model found was 0.008388.

Figure 3.11 illustrates the results of this second exploration. As it can be seen in Figure 3.11a, the MSE decreases for higher *n_estimators* so exploring higher values might be promising. Also, values of *min_samples_leaf* close to 20 produced again the best results

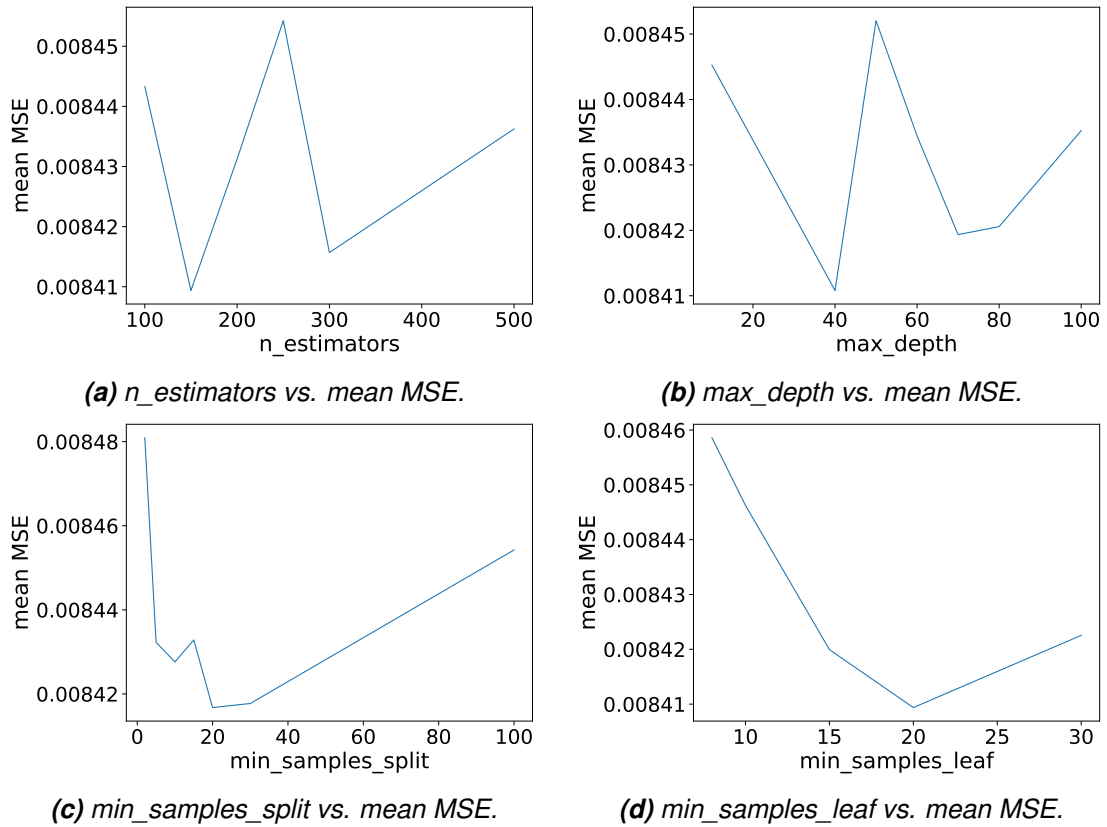


Figure 3.10. First search results.

Table 3.8. Second search hyperparameter values. Random Search with 100 iterations.

Identifier	Hyperparameter	Value Ranges	Best Found
1	<i>n_estimators</i>	100, 150, 200, 250, 300, 500, 1000	1000
2	<i>max_depth</i>	10, 20, 30, 40, 50, 60, 70, 80, 90, 100, None	90
3	<i>min_samples_split</i>	5, 10, 15, 20, 25, 30, 50	20
4	<i>min_samples_leaf</i>	10, 15, 20, 25, 30	20
5	<i>max_features</i>	0.33	0.33
6	<i>bootstrap</i>	True	True
mean MSE			0.008388

as shown in Figure 3.11d. However, *max_depth* and *min_samples_split* did not show clear patterns for the values explored.

Therefore, a third Random Search was run narrowing the values of *min_samples_split* and *min_samples_leaf*. In addition, a greater number of *n_estimators* was tested. The hyperparameter values explored are listed in Table 3.9.

As can be seen, the best model obtained did not improve results from the second Random Search. Therefore, no further important improvements seem possible and the hyperparameter optimization can conclude. A summary of the whole process is listed in Table 3.10.

As shown, the model selected was the best obtained in the second search. Therefore,

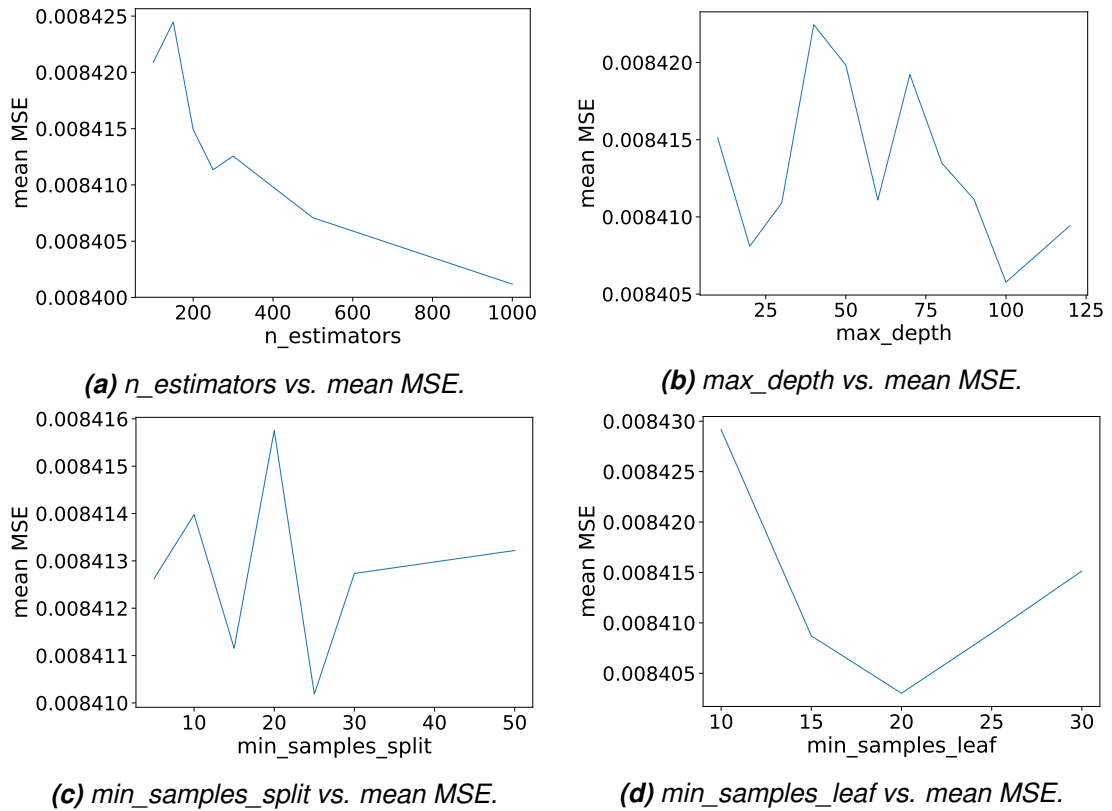


Figure 3.11. Second search results.

Table 3.9. Third search hyperparameter values. Random Search with 100 iterations.

Identifier	Hyperparameter	Value Ranges	Best Found
1	$n_estimators$	500, 1000, 2000, 5000	5000
2	max_depth	10, 20, 30, 40, 50, 60, 70, 80, 90, 100, None	30
3	$min_samples_split$	5, 10, 15, 20, 25	5
4	$min_samples_leaf$	16, 18, 20, 22, 24	20
5	$max_features$	0.33	0.33
6	$bootstrap$	True	True
mean MSE			0.008389

Table 3.10. Summary of Random Forest hyperparameter optimization process for day ahead forecast.

Model	mean MSE	Improvement (%)
Base model	0.008877	—
First search	0.008404	5.33
Second search	0.008388	5.51
Third search	0.008389	5.50
Final	0.008388	5.51

the prediction error improved 5.51% after hyperparameter optimization.

3.6 Feed-Forward Neural Network Experiments

3.6.1 Next Hour Forecast

The next algorithm tested was the Feed-Forward Neural Network, which is the simplest neural network type. Several Random Searches were run to find the optimal hyperparameters, as done with Random Forest. The Neural Network hyperparameters explored were:

1. *neurons_1*: the number of neurons in the first hidden layer.
2. *neurons_2*: the number of neurons in the second hidden layer.
3. *third_layer*: whether to add a third hidden layer.
4. *neurons_3*: the number of neurons in the third hidden layer. Only considered when there is a third hidden layer.
5. *output_activation_function*: the activation function in the output layer. Linear is the common choice for regression problems.
6. *hidden_activation_function*: the activation function in the hidden layers. ReLU is the mostly used, although there are others as the sigmoid or the hyperbolic tangent.
7. *optimizer*: the algorithm used to update the weights in order to minimize the error score. Common optimizers are Scaled Gradient Descent (SGD), Adam and Root Mean Square prop (RMSprop).
8. *lr*: the learning rate of the optimizer.
9. *momentum*: parameter that accelerates SGD in the relevant direction and dampens oscillations. Only considered when the optimizer is SGD.
10. *nesterov*: whether to apply Nesterov momentum.
11. *epochs*: the number of epochs. It defines the number of times that the model will train on the entire training set. Increasing this number increases the training time.
12. *batch_size*: the number of samples used to train before updating the weights. Lower values require more training time. Common used values are powers of two.
13. *dropout*: the dropout rate. Dropout consists on dropping randomly a defined number of neurons of the network. Thus, the connections of the dropped neurons are ignored. It is a regularization technique used to prevent overfitting.
14. *l2*: L2 weight regularization. It adds a penalty for weight size to the loss function. Used to improve generalization and prevent overfitting.
15. *weight_constraint*: it limits the size of the weights in order to avoid overfitting by forcing the weight vector to satisfy $\|\vec{w}\|_2 < c_w$, where \vec{w} is the weight vector of every

neuron and c_w is the weight constraint.

16. *kernel_initializer*: the way the initial random weights are set. He (He-et-al) initialization is often used with ReLU activation function and Xavier initialization with tanh.

As it can be seen, there is a greater amount of hyperparameters to optimize than in Random Forest, which adds more complexity to the optimization process. Moreover, some of these hyperparameters might interact between them. In order to simplify calculations, the structure of the Neural Network is fixed to two hidden layers with 100 neurons each. Thus, the rest of hyperparameters as the optimizer and the regularization methods were optimized first for this fixed structure. Then, using the found optimal hyperparameters, different Neural Network structures were explored to find the final model. The activation functions were also fixed to be linear in the output layer and ReLU in the hidden layers.

First, a base Feed Forward Neural Network model was created with the hyperparameters shown in Table 3.11 and tested on 3 different training and validation splits. The average MSE obtained was 0.003696.

Table 3.11. Base model hyperparameters and MSE.

Identifier	Hyperparameter	Value
1	<i>neurons_1</i>	100
2	<i>neurons_2</i>	100
3	<i>third_layer</i>	False
4	<i>neurons_3</i>	—
5	<i>output_activation</i>	'linear'
6	<i>hidden_activation</i>	'relu'
7	<i>optimizer</i>	SGD
8	<i>lr</i>	0.01
9	momentum	0
10	nesterov	False
11	epochs	50
12	batch_size	64
13	dropout	—
14	l2	—
15	weight_constraint	—
16	kernel_initializer	'he_normal'
mean MSE		0.003696

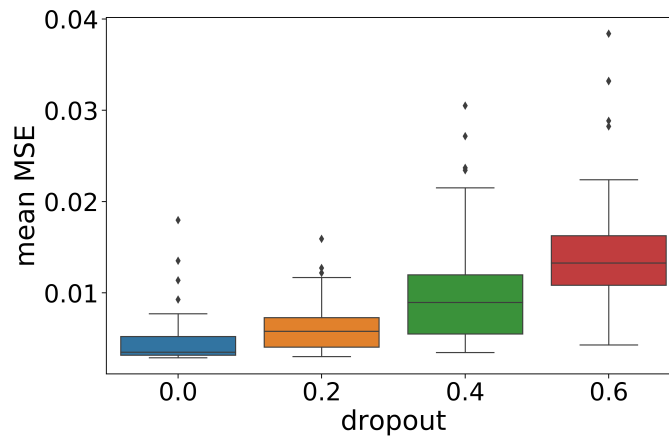
Then, the first Random Search was run exploring the hyperparameter ranges shown in Table 3.12. A total of 150 iterations were run on 3 training and validation splits each.

The lowest MSE obtained was 0.002907, which improved 21.35% the base model. The strongest relationship was that dropout rates higher than 0.2 often produced poor results as shown in Figure 3.12. For that reason, it was decided to study only data with dropout rates of 0 and 0.2, as higher dropout rates could difficult the results interpretation for the

Table 3.12. First search hyperparameter values. Random Search with 150 iterations.

Identifier	Hyperparameter	Range values	Best found
1	<i>neurons_1</i>	100	100
2	<i>neurons_2</i>	100	100
3	<i>third_layer</i>	False	False
4	<i>neurons_3</i>	50	—
5	<i>output_activation</i>	'linear'	'linear'
6	<i>hidden_activation</i>	'relu'	'relu'
7	<i>optimizer</i>	SGD, Adam, RMSprop	RMSprop
8	<i>lr</i>	0.0001, 0.001, 0.01	0.001
9	<i>momentum</i>	0, 0.9	—
10	<i>nesterov</i>	True, False	—
11	<i>epochs</i>	10, 20, 50	50
12	<i>batch_size</i>	32, 64, 128	64
13	<i>dropout</i>	0, 0.2, 0.4, 0.6	0
14	<i>l2</i>	0.00001, 0.0001, 0.001	0.00001
15	<i>weight_constraint</i>	1, 2, 3, 4, 5	1
16	<i>kernel_initializer</i>	'he_normal', 'he_uniform'	'he_normal'
mean MSE			0.002907

rest of hyperparameters.

**Figure 3.12.** Relationship between dropout_rate and mean MSE.

Then, the different optimizers and learning rates were studied. As shown in Figure 3.13, Adam and RMSprop produced the best results, while SGD only obtained decent performance with a learning rate of 0.01. Then, an hypothesis was made. SGD is an algorithm that converges slower than Adam and RMSprop to the optimal solution. Thus, it might be possible that for the number of epochs tested, the experiments using SGD optimizer did not have time to converge. Moreover, a higher number of training epochs is usually required when using weight regularization methods. In order to verify the hypothesis, three models using the three different optimizers along with l2 weight regularization were

created and their loss was tracked during the whole training as shown in Figure 3.14.

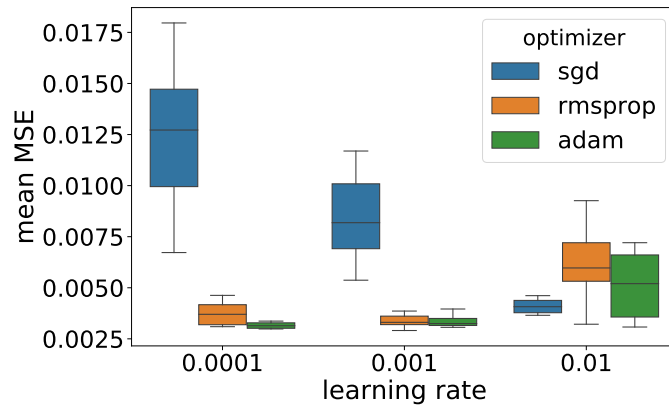


Figure 3.13. Relationship between optimizers, learning rate and MSE.

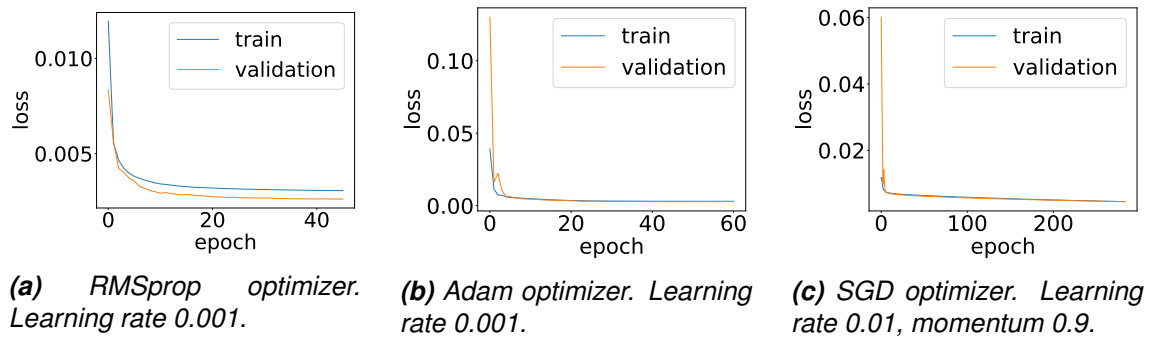


Figure 3.14. Training and validation loss over epochs for different optimizers.

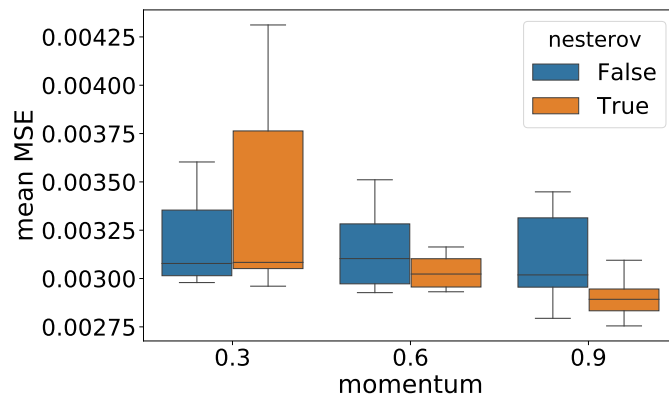
The models were trained until the validation loss stopped decreasing during 20 consecutive epochs. The train and validation losses shown are not just the prediction MSE. When using weight regularization methods, a penalty for large weights is added to the loss function used to optimize the model during training. Therefore, the used loss is the sum of the prediction loss and the weight penalty. As shown, RMSprop and Adam converge fast, requiring less than 50 epochs. However, even with a higher learning rate and a high momentum parameter added, SGD still converges considerably slower. For that reason, it was decided to run a separate Random Search for SGD optimizer to check if it can beat the rest of the optimizers when it converges. Therefore, 50 iterations with the grid of hyperparameters shown in Table 3.13 were run.

In these new experiments SGD produced better results than the previous ones and the lowest obtained MSE was 0.002755. The configurations that performed best were those with *momentum 0.9* and *nesterov True* as shown in Figure 3.15. *l2* weight regularization with value 0.0001 also produced better results. The rest of the hyperparameters did not show a clear pattern. For that reason, a third Random Search of 15 iterations was run, narrowing the hyperparameter ranges to the mentioned values as shown in Table 3.14.

Later, with the optimal configuration found, 50 experiments were run to find the optimal structure of the network exploring the hyperparameters listed on Table 3.15.

Table 3.13. Second search hyperparameter values. Random Search with 50 iterations.

Identifier	Hyperparameter	Range values	Best found
1	<i>neurons_1</i>	100	100
2	<i>neurons_2</i>	100	100
3	<i>third_layer</i>	False	False
4	<i>neurons_3</i>	—	—
5	<i>output_activation</i>	'linear'	'linear'
6	<i>hidden_activation</i>	'relu'	'relu'
7	<i>optimizer</i>	SGD	SGD
8	<i>lr</i>	0.01	0.01
9	<i>momentum</i>	0, 0.3, 0.6, 0.9	0.9
10	<i>nesterov</i>	True, False	True
11	<i>epochs</i>	250	250
12	<i>batch_size</i>	64, 128	64
13	<i>dropout</i>	0, 0.2	0.2
14	<i>l2</i>	0.00001, 0.0001, 0.001	0.0001
15	<i>weight_constraint</i>	1, 3, 5	1
16	<i>kernel_initializer</i>	'he_normal', 'he_uniform'	'he_uniform'
mean MSE			0.002755

**Figure 3.15.** Relationship between momentum, nesterov and MSE.

The best configuration found was a two hidden layer network with 75 and 150 neurons. The MSE obtained was 0.002738, which did not improve considerably the previous results. Moreover, no clear relationship between the structure and the error was found. Therefore, the best configuration obtained in this fourth Random Search was selected as the final configuration. A summary of the optimization process is shown in Table 3.16.

Finally, the definitive model was built with the optimal configuration found using the early stopping technique. Early stopping consists on stopping the training when the validation loss stops decreasing for a determined number of epochs. Thus, the training can be stopped when no more important improvements can be achieved or when the validation

Table 3.14. Third search hyperparameter values. Random Search with 15 iterations.

Identifier	Hyperparameter	Range values	Best found
1	<i>neurons_1</i>	100	100
2	<i>neurons_2</i>	100	100
3	<i>third_layer</i>	False	False
4	<i>neurons_3</i>	—	—
5	<i>output_activation</i>	'linear'	'linear'
6	<i>hidden_activation</i>	'relu'	'relu'
7	<i>optimizer</i>	SGD	SGD
8	<i>lr</i>	0.01	0.01
9	<i>momentum</i>	0.9	0.9
10	<i>nesterov</i>	True	True
11	<i>epochs</i>	250	250
12	<i>batch_size</i>	64, 128	128
13	<i>dropout</i>	0, 0.2	0
14	<i>l2</i>	0.0001	0.0001
15	<i>weight_constraint</i>	1, 3, 5	1
16	<i>kernel_initializer</i>	'he_normal', 'he_uniform'	'he_uniform'
mean MSE			0.002740

Table 3.15. Fourth search hyperparameter values. Random Search with 50 iterations.

Identifier	Hyperparameter	Range values	Best found
1	<i>neurons_1</i>	25, 50, 75, 100, 125, 150	75
2	<i>neurons_2</i>	25, 50, 75, 100, 125, 150	150
3	<i>third_layer</i>	True, False	False
4	<i>neurons_3</i>	25, 50, 75, 100, 125, 150	—
5	<i>output_activation</i>	'linear'	'linear'
6	<i>hidden_activation</i>	'relu'	'relu'
7	<i>optimizer</i>	SGD	SGD
8	<i>lr</i>	0.01	0.01
9	<i>momentum</i>	0.9	0.9
10	<i>nesterov</i>	True	True
11	<i>epochs</i>	250	250
12	<i>batch_size</i>	128	128
13	<i>dropout</i>	0	0
14	<i>l2</i>	0.0001	0.0001
15	<i>weight_constraint</i>	1	1
16	<i>kernel_initializer</i>	'he_uniform'	'he_uniform'
mean MSE			0.002738

Table 3.16. Summary of Feedforward Neural Network hyperparameter optimization process for next hour forecast.

Model	meanMSE	Improvement (%)
Base model	0.003696	—
First search	0.002907	21.35
Second search	0.002755	25.46
Third search	0.002740	25.87
Fourth search	0.002738	25.92
Final	0.002738	25.92

loss starts to increase due to overfitting. The training was stopped when the loss stopped decreasing for 50 consecutive epochs.

Then, after stopping the training, the best epoch weights are loaded and set as final weight values. In other words, the final weights are loaded from the epoch that achieved the lowest validation error.

Early stopping was not used along with Grid Search and Random Search because the programming code did not permit to integrate them together. However, a sufficient number of epochs was used in order to allow the trained models to converge to a solution. Therefore, early stopping was only used during the training of the final model.

3.6.2 Day Ahead Forecast

Similarly to the process done for next hour forecast, several configurations were tested in order to find a model that could predict the day ahead observations. However, previous results from next hour forecast hyperparameter optimization were used to narrow the values to explore. Thus, the optimizer was fixed to SGD with *momentum* 0.9 and *nesterov* True, while the dropout rate was set to 0. Moreover, the first experiments were tested on a neural network structure of two hidden layers with 200 neurons each. In addition, the batch size was set to 64 and the number of epochs to 300.

The number of epochs needed by the model to converge to a solution depend on the batch size. Therefore, the number of epochs was chosen so that the error of a base model was able to converge for a batch size of 64 as shown in Figure 3.16. This base model was trained and tested on three folds using the hyperparameter values listed in Table 3.17.

Then, a first Grid Search was run for the hyperparameter values listed in Table 3.18. All configurations were tested instead of a smaller random selection as it was considered that the time required to calculate 32 configurations would not be excessive.

The MSE obtained by the best model was 0.008518. Also, l2 weight regularization 0.0001 clearly showed the best results while small values of weight_constraint seemed to get

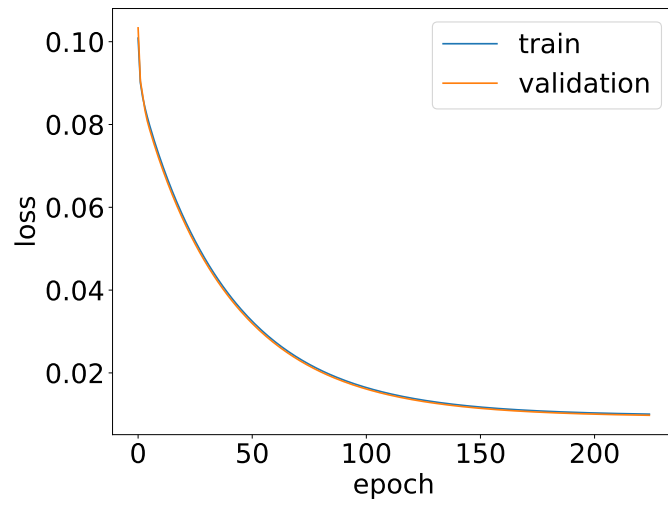


Figure 3.16. Loss evolution through epochs for a 2 hidden layers neural network and batch size 64.

Table 3.17. Base model hyperparameters and MSE.

Identifier	Hyperparameter	Value
1	<i>neurons_1</i>	200
2	<i>neurons_2</i>	200
3	<i>third_layer</i>	False
4	<i>neurons_3</i>	—
5	<i>output_activation</i>	'linear'
6	<i>hidden_activation</i>	'relu'
7	<i>optimizer</i>	SGD
8	<i>lr</i>	0.01
9	<i>momentum</i>	0.9
10	<i>nesterov</i>	True
11	<i>epochs</i>	300
12	<i>batch_size</i>	64
13	<i>dropout</i>	—
14	<i>l2</i>	—
15	<i>weight_constraint</i>	—
16	<i>kernel_initializer</i>	'he_uniform'
mean MSE		0.008711

Table 3.18. First search hyperparameter values. Grid Search with 32 iterations.

Identifier	Hyperparameter	Range values	Best found
1	<i>neurons_1</i>	200	200
2	<i>neurons_2</i>	200	200
3	<i>third_layer</i>	False	False
4	<i>neurons_3</i>	—	—
5	<i>output_activation</i>	'linear'	'linear'
6	<i>hidden_activation</i>	'relu'	'relu'
7	<i>optimizer</i>	SGD	SGD
8	<i>lr</i>	0.01	0.01
9	<i>momentum</i>	0.9	0.9
10	<i>nesterov</i>	True	True
11	<i>epochs</i>	300	300
12	<i>batch_size</i>	64	64
13	<i>dropout</i>	0	0
14	<i>l2</i>	0, 0.00001, 0.0001, 0.001	0.0001
15	<i>weight_constraint</i>	1, 2, 3, 4	2
16	<i>kernel_initializer</i>	'he_uniform', 'he_normal'	'he_normal'
mean MSE			0.008518

lower error as shown in Figure 3.17. On the contrary, the *kernel_initializer* did not show a clear relationship.

Then, the optimal structure was explored using the best hyperparameter values found in the first Grid Search. Thus, a second Grid Search was run to study network structures with two hidden layers testing different numbers of neurons. Structures with three hidden layers will be studied later to check if results improve after adding a third hidden layer. The values used during the second Grid Search and the best found configuration are listed in Table 3.19.

The obtained results indicate that increasing the number of neurons of the network reduce the prediction error. However, this reduction becomes negligible when the number of neurons is very high as shown in Figure 3.18. Also, it does not seem to matter which layer has more neurons. What seems to be important is the total number of neurons in the network. The darkest area corresponding to structures with lowest error, contains structures with 900 neurons or greater. The best structure found had 400 neurons in the first hidden layer, 500 neurons in the second and had a MSE of 0.008478.

Then, a Random Search of 50 iterations was run to study structures with 3 hidden layers. The values explored are listed in Table 3.20.

The best configuration produced a MSE of 0.008490. As it can be seen, adding a third hidden layer did not improve previous results. Therefore, the structure chosen was 2

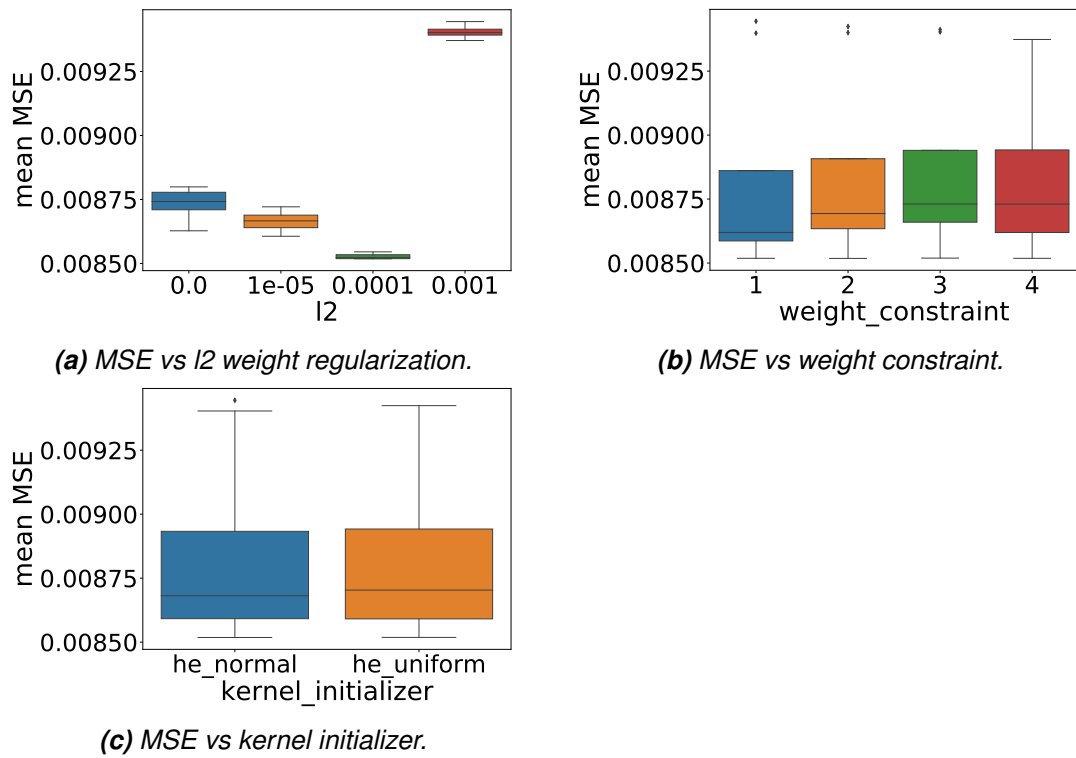


Figure 3.17. First search results.

Table 3.19. Second search hyperparameter values. Grid Search with 49 iterations.

Identifier	Hyperparameter	Range values	Best found
1	<i>neurons_1</i>	50, 100, 200, 300, 400, 500, 600	400
2	<i>neurons_2</i>	50, 100, 200, 300, 400, 500, 600	500
3	<i>third_layer</i>	False	False
4	<i>neurons_3</i>	—	—
5	<i>output_activation</i>	'linear'	'linear'
6	<i>hidden_activation</i>	'relu'	'relu'
7	<i>optimizer</i>	SGD	SGD
8	<i>lr</i>	0.01	0.01
9	<i>momentum</i>	0.9	0.9
10	<i>nesterov</i>	True	True
11	<i>epochs</i>	300	300
12	<i>batch_size</i>	64	64
13	<i>dropout</i>	0	0
14	<i>l2</i>	0.0001	0.0001
15	<i>weight_constraint</i>	2	2
16	<i>kernel_initializer</i>	'he_normal'	'he_normal'
mean MSE			0.008478

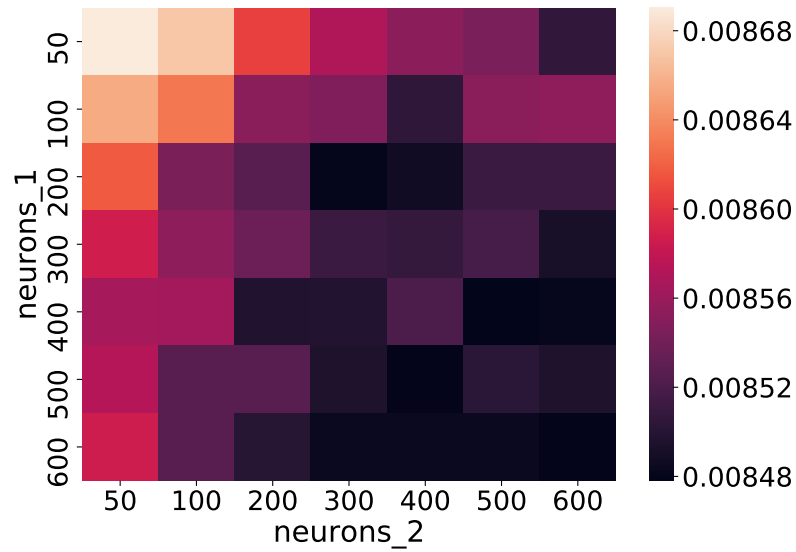


Figure 3.18. Mean MSE for different number of neurons of two hidden layer network structures.

Table 3.20. Third search hyperparameter values. Random Search with 50 iterations.

Identifier	Hyperparameter	Range values	Best found
1	<i>neurons_1</i>	200, 210, 220, ..., 600	260
2	<i>neurons_2</i>	200, 210, 220, ..., 600	550
3	<i>third_layer</i>	True	True
4	<i>neurons_3</i>	200, 210, 220, ..., 600	580
5	<i>output_activation</i>	'linear'	'linear'
6	<i>hidden_activation</i>	'relu'	'relu'
7	<i>optimizer</i>	SGD	SGD
8	<i>lr</i>	0.01	0.01
9	<i>momentum</i>	0.9	0.9
10	<i>nesterov</i>	True	True
11	<i>epochs</i>	300	300
12	<i>batch_size</i>	64	64
13	<i>dropout</i>	0	0
14	<i>l2</i>	0.0001	0.0001
15	<i>weight_constraint</i>	2	2
16	<i>kernel_initializer</i>	'he_normal'	'he_normal'
mean MSE			0.008490

hidden layers with 400 and 500 neurons in the first and second hidden layer respectively. Table 3.21 show a summary of the whole hyperparameter optimization process.

Table 3.21. Summary of Feedforward Neural Network hyperparameter optimization process for day ahead forecast.

Model	MSE	Improvement (%)
Base model	0.008711	—
First Random Search	0.008518	2.22
Second Random Search	0.008478	2.67
Third Random Search	0.008490	2.54
Final	0.008478	2.67

Finally, the model with the optimal configuration was trained using also the early stopping technique and was tested on the validation test. The obtained results can be found in Chapter 4.

3.7 Long Short-Term Memory Network Experiments

3.7.1 Data structure

LSTMs need the input data shape to have 3 dimensions as illustrated in Figure 3.19: samples, time steps and features. For that reason, instead of using only the last observation available for time and day features, this time, the whole vector of last observations will be fed into the LSTM.

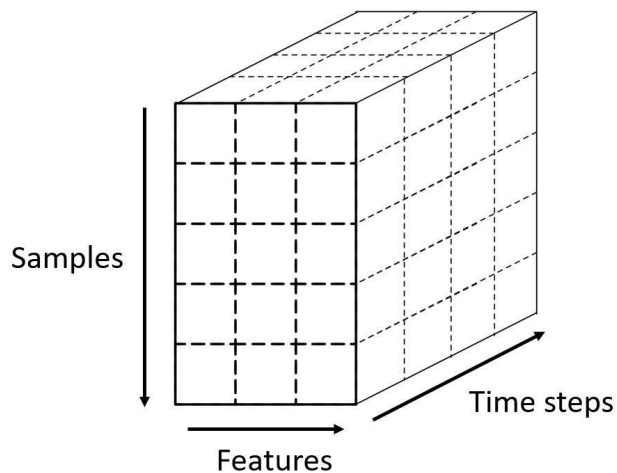


Figure 3.19. 3-dimensional shape of data fed to LSTM.

Moreover, two different lengths of last observations will be studied: last 24 and last 48 hours. Longer sequences would increase considerably the training time, which is already much longer than the required in FNNs. For that reason the maximum sequence length studied will be 48 observations.

3.7.2 Next Hour Forecast

LSTMs and FNNs are considerably different in structure but most of the important hyperparameters are the same. In fact, the hyperparameters tuned during LSTM hyperparameter optimization were the same hyperparameters tuned during FNN hyperparameter optimization. However, the input sequence length was also studied as explained above.

First, a base model with the values listed in Table 3.22 was studied. The mean MSE obtained was 0.002659.

Table 3.22. Base model hyperparameters and mean MSE.

Identifier	Hyperparameter	Value
1	<i>neurons_1</i>	100
2	<i>neurons_2</i>	100
3	<i>third_layer</i>	False
4	<i>neurons_3</i>	—
5	<i>output_activation</i>	'linear'
6	<i>hidden_activation</i>	'relu'
7	<i>optimizer</i>	Adam
8	<i>lr</i>	0.001
9	momentum	—
10	nesterov	—
11	epochs	50
12	batch_size	128
13	dropout	—
14	l2	—
15	weight_constraint	—
16	kernel_initializer	'glorot_uniform'
17	input sequence length	24
mean MSE		0.002753

Training LSTMs require a considerably greater amount of time than the previous algorithms. For that reason, fewer iterations were run during hyperparameter optimization and some of the hyperparameters were fixed to common values. The selected optimizer was Adam which is a popular choice because it usually produces good results fast. Then, the learning rate was set to 0.001 which is the common recommendation for Adam optimizer. Linear activation function was used in the output layer and ReLU after each hidden layer. Finally, the batch size was set to 128. A large value of batch size was chosen in order to accelerate training.

Then, a Grid Search was run to decide first whether to input the last 24 or 48 hourly observations. 4 different LSTM structures were explored for each option to select the best choice independently of the LSTM structure. The rest of hyperparameter values

were the same as the base model. A total number of 8 configurations were explored. The validation error for each fold and the average value are listed in Table 3.23.

Table 3.23. First search results. Grid Search with 8 iterations.

Input sequence length	<i>neurons_1</i>	<i>neurons_2</i>	MSE 0	MSE 1	MSE 2	mean MSE
24	25	25	0.003177	0.003621	0.001866	0.002888
24	50	50	0.002965	0.003402	0.001836	0.002734
24	100	100	0.002831	0.003442	0.001987	0.002753
24	200	200	0.002902	0.003371	0.001979	0.002751
48	25	25	0.002944	0.003548	0.001912	0.002801
48	50	50	0.003033	0.003506	0.001920	0.002819
48	100	100	0.002744	0.003366	0.001866	0.002658
48	200	200	0.003228	0.003438	0.001931	0.002865

Remember that during Grid Search the data is divided into 4 splits in order to create 3 train and validation folds. In Python language these are defined as fold 0, 1 and 2. Thus, fold 0 uses split 1 for train and split 2 for validation, while fold 1 uses splits 1 and 2 for train and split 3 for validation and fold 2 uses splits 1, 2 and 3 for train and split 4 for validation.

The variation of the validation error between folds is not only due to the influence of training the LSTM with more or less data. It is also influenced by the magnitude of the validation data. GHI is greater during summer than during winter. Thus, if a fold has more summer samples in the validation set than other fold, it is likely to have larger error. Therefore, the validation error will not be compared between folds in order to study whether using more training data improves the prediction results, although it is expected that it does. What will be studied in some cases is how a hyperparameter affects the prediction error when a large amount of training data is used as in Fold 2. The reason is that the final model will be trained with even more training data than Fold 2. Moreover, LSTMs usually need a large amount of data to achieve satisfactory results.

As it can be seen, the best result is for sequence length 48. However, when looking at the overall there is not a clear difference in performance between the two sequence lengths. Moreover, the calculated error has some variance as results can vary from different models even if they use the same hyperparameter configuration. Therefore, the next explored configurations will use sequence length 24 as it requires less time to train and produces similar results.

Then a Grid Search was run to study if implementing dropout, weight constraints or changing the weight initialization produce better models. L2 weight regularization will be studied in a separate Grid Search as it usually requires more epochs to converge. A total of 8 configurations were explored with the values listed in Table 3.24.

The lowest MSE was 0.002738. The hyperparameter that affected most the error score

Table 3.24. Second search hyperparameter values. Grid Search with 8 iterations.

Identifier	Hyperparameter	Range values	Best found
1	<i>neurons_1</i>	50	50
2	<i>neurons_2</i>	50	50
3	<i>third_layer</i>	False	False
4	<i>neurons_3</i>	—	—
5	<i>output_activation</i>	'linear'	'linear'
6	<i>hidden_activation</i>	'relu'	'relu'
7	<i>optimizer</i>	Adam	Adam
8	<i>lr</i>	0.001	0.001
9	<i>momentum</i>	—	—
10	<i>nesterov</i>	—	—
11	<i>epochs</i>	25, 50	50
12	<i>batch_size</i>	128	128
13	<i>dropout</i>	0, 0.2	0
14	<i>l2</i>	0	0
15	<i>weight_constraint</i>	None, 2	2
16	<i>kernel_initializer</i>	'glorot_uniform', 'he_uniform'	'glorot_uniform'
17	input sequence length	24	24
mean MSE			0.002738

was the *kernel_initializer*. As illustrated in Figure 3.20d, changing the weight initialization to He increased the error despite it is commonly recommended when ReLU is used as activation function. The same happened when adding dropout rate, which produced worse results as shown in Figure 3.20b. The benefits of adding a weight constraint are not clear while models trained with 50 epochs performed better than models with 25 as shown in Figure 3.20a.

As said above, l2 weight regularization was studied in a third Grid Search. The results of the second Grid Search were used to narrow the values to explore. Therefore, *kernel_initializer* and *dropout* were set to 'glorot_uniform' and 0 respectively. Configurations with and without weight constraint were studied again while the number of epochs was set to 100 as models trained with weight regularization usually require more epochs to converge. A total of 8 iterations were run with the values listed in Table 3.25.

Results slightly improved after adding l2 weight regularization and the best model produced a MSE of 0.002687. Figure 3.21a illustrates the error for different l2 values including some samples from the second Grid Search without l2 weight regularization (l2 0). These were samples that satisfied *kernel_initializer* 'glorot_uniform', *dropout* 0 and *epochs* 50. As it can be seen the optimal l2 weight regularization value is 10^{-5} which produced slightly better results than l2 0. Moreover, this time adding a weight constraint clearly reduced the prediction error as shown in Figure 3.21b where 0.0 means no weight

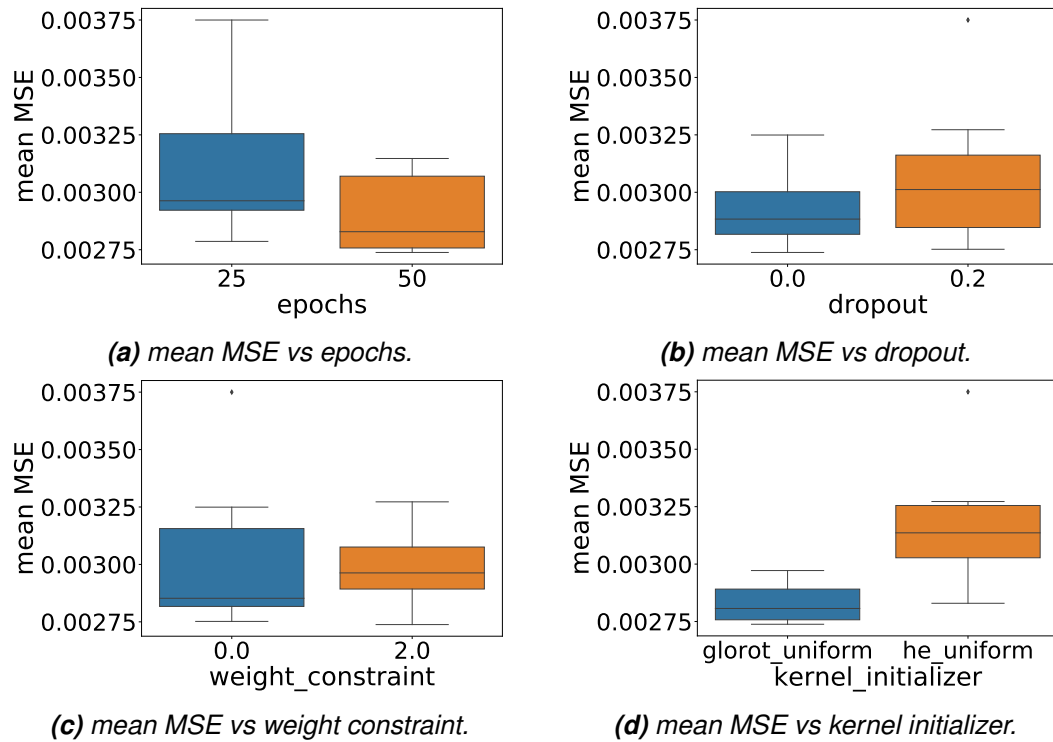


Figure 3.20. First Grid Search Results.

Table 3.25. Third search hyperparameter values. Grid Search with 8 iterations.

Identifier	Hyperparameter	Range values	Best found
1	<i>neurons_1</i>	50	50
2	<i>neurons_2</i>	50	50
3	<i>third_layer</i>	False	False
4	<i>neurons_3</i>	—	—
5	<i>output_activation</i>	'linear'	'linear'
6	<i>hidden_activation</i>	'relu'	'relu'
7	<i>optimizer</i>	Adam	Adam
8	<i>lr</i>	0.001	0.001
9	<i>momentum</i>	—	—
10	<i>nesterov</i>	—	—
11	<i>epochs</i>	100	100
12	<i>batch_size</i>	128	128
13	<i>dropout</i>	0	0
14	<i>l2</i>	$10^{-6}, 10^{-5}, 10^{-4}, 10^{-3}$	10^{-5}
15	<i>weight_constraint</i>	None, 2	2
16	<i>kernel_initializer</i>	'glorot_uniform'	'glorot_uniform'
17	input sequence length	24	24
mean MSE			0.002687

constraint.

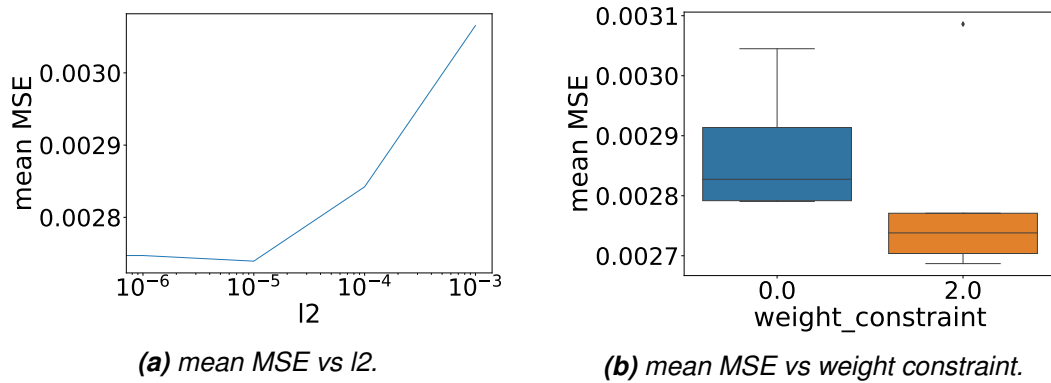


Figure 3.21. Second Grid Search Results.

Then, a fourth Grid Search was run to find the optimal value for *weight_constraint*. In addition, 0 and 10^{-5} l_2 values were explored to confirm whether l_2 weight regularization reduces the prediction error. 12 iterations were run in total with the values listed in Table 3.26.

Table 3.26. Fourth search hyperparameter values. Grid Search with 12 iterations.

Identifier	Hyperparameter	Range values	Best found
1	<i>neurons_1</i>	50	50
2	<i>neurons_2</i>	50	50
3	<i>third_layer</i>	False	False
4	<i>neurons_3</i>	—	—
5	<i>output_activation</i>	'linear'	'linear'
6	<i>hidden_activation</i>	'relu'	'relu'
7	<i>optimizer</i>	Adam	Adam
8	<i>lr</i>	0.001	0.001
9	<i>momentum</i>	—	—
10	<i>nesterov</i>	—	—
11	<i>epochs</i>	50, 100	100
12	<i>batch_size</i>	128	128
13	<i>dropout</i>	0	0
14	<i>l2</i>	0, 10^{-5}	0
15	<i>weight_constraint</i>	1, 3, 4	4
16	<i>kernel_initializer</i>	'glorot_uniform'	'glorot_uniform'
17	input sequence length	24	24
mean MSE			0.002702

As it can be seen, *weight_constraint* 2 was not added to this Grid Search in order to avoid repeating calculations that would increase unnecessarily the calculation time. Thus, previous samples with *weight_constraint* None and 2 and l_2 0 and 10^{-5} were added later to the third Grid Search results to compare them. The best value found for *weight_constraint*

was 2 as illustrated in Figure 3.22.

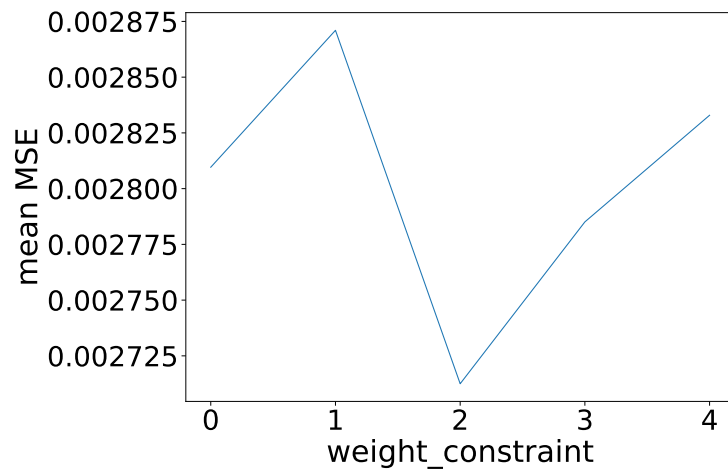


Figure 3.22. MSE vs weight constraint

In addition, models trained for 100 epochs performed better than those trained for 50 as shown in Figure 3.23. It can also be seen that despite l_2 0 produced better results in average, l_2 10^{-5} performed better when training with larger amount of data. Therefore, $weight_constraint$ 2 and l_2 10^{-5} were selected for future searches.

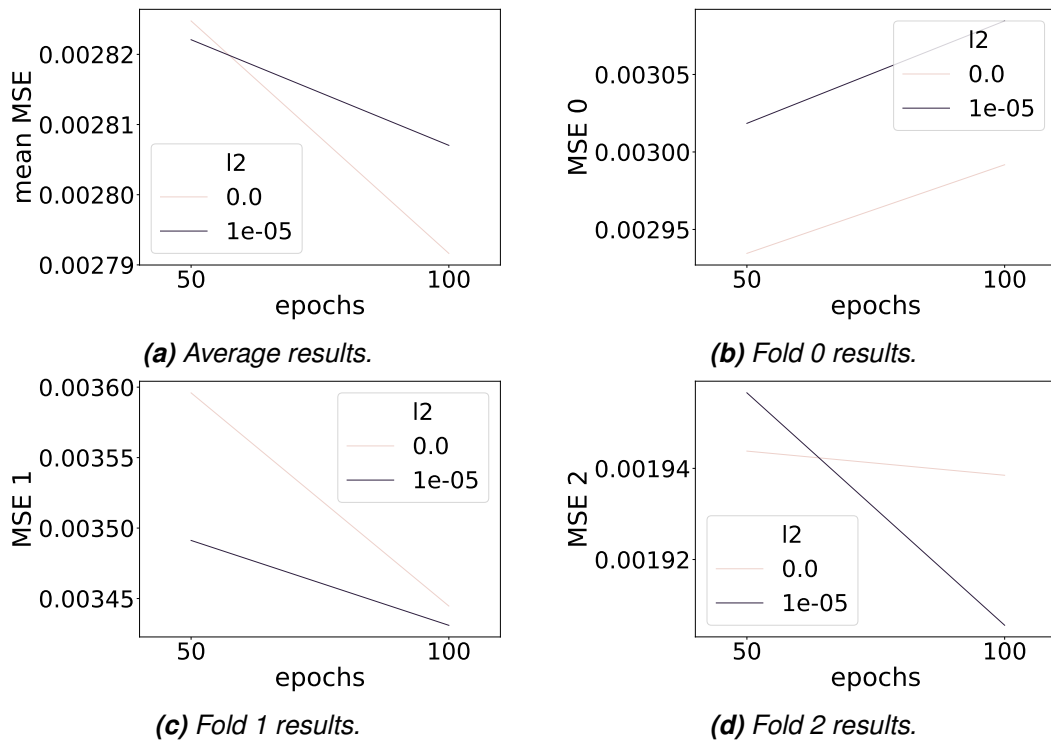


Figure 3.23. Fourth Grid Search results.

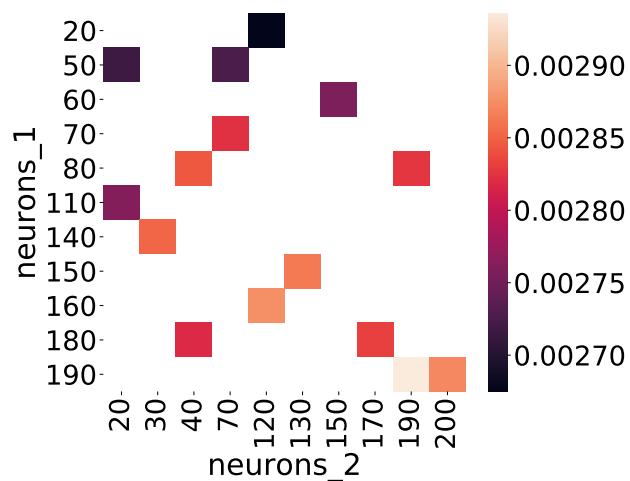
Then, the optimal number of neurons for each layer was explored running a Random Search of 15 iterations with the values listed in Table 3.27.

The mean validation errors produced by each explored configuration are illustrated in a heat map in Figure 3.24. Thus, the vertical axis represents the number of neurons in

Table 3.27. Fifth search hyperparameter values. Random Search with 15 iterations.

Identifier	Hyperparameter	Range values	Best found
1	<i>neurons_1</i>	20, 30, 40, ..., 200	20
2	<i>neurons_2</i>	20, 30, 40, ..., 200	120
3	<i>third_layer</i>	False	False
4	<i>neurons_3</i>	—	—
5	<i>output_activation</i>	'linear'	'linear'
6	<i>hidden_activation</i>	'relu'	'relu'
7	<i>optimizer</i>	Adam	Adam
8	<i>lr</i>	0.001	0.001
9	<i>momentum</i>	—	—
10	<i>nesterov</i>	—	—
11	<i>epochs</i>	100	100
12	<i>batch_size</i>	128	128
13	<i>dropout</i>	0	0
14	<i>l2</i>	10^{-5}	10^{-5}
15	<i>weight_constraint</i>	2	2
16	<i>kernel_initializer</i>	'glorot_uniform'	'glorot_uniform'
17	input sequence length	24	24
mean MSE			0.002675

the first hidden layer while the horizontal axis represents the number of neurons in the second hidden layer.

**Figure 3.24.** Mean MSE produced by each structure configuration explored in fifth search.

Generally, models with a large number of neurons performed worse than those with fewer. For that reason it was decided to run a last Random Search narrowing the search to smaller numbers of neurons. Thus, 15 additional iterations were run exploring the values listed in Table 3.28.

Table 3.28. Sixth search hyperparameter values. Random Search with 15 iterations.

Identifier	Hyperparameter	Range values	Best found
1	<i>neurons_1</i>	20, 30, 40, ..., 80	40
2	<i>neurons_2</i>	20, 30, 40, ..., 120	20
3	<i>third_layer</i>	False	False
4	<i>neurons_3</i>	—	—
5	<i>output_activation</i>	'linear'	'linear'
6	<i>hidden_activation</i>	'relu'	'relu'
7	<i>optimizer</i>	Adam	Adam
8	<i>lr</i>	0.001	0.001
9	<i>momentum</i>	—	—
10	<i>nesterov</i>	—	—
11	<i>epochs</i>	100	100
12	<i>batch_size</i>	128	128
13	<i>dropout</i>	0	0
14	<i>l2</i>	10^{-5}	10^{-5}
15	<i>weight_constraint</i>	2	2
16	<i>kernel_initializer</i>	'glorot_uniform'	'glorot_uniform'
17	input sequence length	24	24
mean MSE			0.002710

The best result was produced by a model with 40 and 20 neurons in the first and second hidden layers respectively as illustrated in Figure 3.25. In this figure, results from fourth and fifth Random Searches are shown together. If a configuration was repeated in both searches, the average MSE value was calculated. That is the case of the previous best configuration with 20 and 120 neurons, which stopped being the best configuration after averaging its previous MSE with the new one. That confirms that there is some variability in results so it is not possible to assert that the model with lowest error is the best. However, if similar models have also a low error, it is then likely to be close to the optimal. As it can be seen in Figure 3.25, the configuration with 40 and 20 neurons in the first and second hidden layers respectively is located in the darkest area which corresponds to models with lowest error. Therefore, it seems a favorable choice.

A summary of the whole optimization process is shown in Table 3.29. As it can be seen, the improvements were not very large compared to the base model.

The final LSTM model was trained using the early stopping technique as done for FNN. Then, the weights from the best epoch were loaded and the model was tested on the validation data. The results and the comparison with the rest of the algorithms are found in Chapter 4.

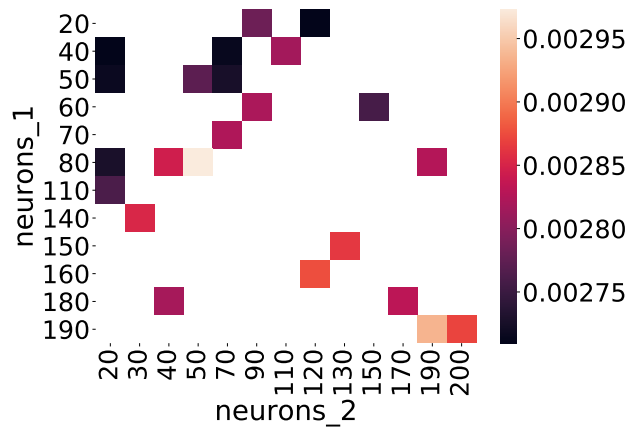


Figure 3.25. Mean MSE produced by each structure configuration explored in fifth and sixth searches.

Table 3.29. Summary of LSTM hyperparameter optimization process for next hour forecast.

Model	mean MSE	Improvement (%)
Base model	0.002753	—
First search	0.002658	3.45
Second search	0.002738	0.54
Third search	0.002687	2.40
Fourth search	0.002702	1.85
Fifth search	0.002675	2.83
Sixth search	0.002710	1.56
Final	0.002710	1.56

3.7.3 Day Ahead Forecast

This time, the LSTM hyperparameters were optimized to create a model that could forecast the hourly day ahead solar irradiance. First, a base model with the hyperparameter values listed in Table 3.30 was trained and tested. The obtained mean validation error was 0.010316.

Then, 4 different LSTM configurations were trained and tested with input sequence lengths 24 and 48. The produced results are listed in Table 3.31.

As it can be seen, the LSTM seems to produce better results when the last 24 observations are input. Instead, an input sequence length of 48 often produced large errors when the number of neurons was large. Therefore, an input sequence length of 24 was selected again. Then, a Grid Search of 8 iterations was run with the hyperparameters listed in Table 3.32.

As illustrated in Figure 3.26a kernel_initializer 'glorot_uniform' or Xavier weight initialization clearly performed better than 'he_uniform'. Also, models without dropout rate

Table 3.30. Base model hyperparameters and mean MSE.

Identifier	Hyperparameter	Value
1	<i>neurons_1</i>	50
2	<i>neurons_2</i>	50
3	<i>third_layer</i>	False
4	<i>neurons_3</i>	—
5	<i>output_activation</i>	'linear'
6	<i>hidden_activation</i>	'relu'
7	<i>optimizer</i>	Adam
8	<i>lr</i>	0.001
9	momentum	0
10	nesterov	False
11	epochs	50
12	batch_size	128
13	dropout	—
14	l2	—
15	weight_constraint	—
16	kernel_initializer	'glorot_uniform'
17	input sequence length	24
mean MSE		0.010316

Table 3.31. First search results. Grid Search with 8 iterations.

Input sequence length	<i>neurons_1</i>	<i>neurons_2</i>	MSE 0	MSE 1	MSE 2	mean MSE
24	25	25	0.010182	0.011363	0.006801	0.009449
24	50	50	0.011310	0.012251	0.007387	0.010316
24	75	75	0.012938	0.013778	0.009029	0.011915
24	100	100	0.011825	0.014389	0.008681	0.011632
48	25	25	0.010931	0.011988	0.006626	0.009848
48	50	50	0.012908	0.012865	0.007979	0.011251
48	75	75	0.014070	0.013561	0.008959	0.012197
48	100	100	0.015622	0.016847	0.009751	0.014073

Table 3.32. Second search hyperparameter values. Grid Search with 8 iterations.

Identifier	Hyperparameter	Range values	Best found
1	<i>neurons_1</i>	25	25
2	<i>neurons_2</i>	25	25
3	<i>third_layer</i>	False	False
4	<i>neurons_3</i>	—	—
5	<i>output_activation</i>	'linear'	'linear'
6	<i>hidden_activation</i>	'relu'	'relu'
7	<i>optimizer</i>	Adam	Adam
8	<i>lr</i>	0.001	0.001
9	<i>momentum</i>	—	—
10	<i>nesterov</i>	—	—
11	<i>epochs</i>	50	50
12	<i>batch_size</i>	128	128
13	<i>dropout</i>	0, 0.2	0
14	<i>l2</i>	0	0
15	<i>weight_constraint</i>	None, 2	None
16	<i>kernel_initializer</i>	'he_uniform', 'glorot_uniform'	'glorot_uniform'
17	input sequence length	24	24
mean MSE			0.009211

performed better as shown in Figure 3.26b. The best result was produced by a model without weight constraint. However, using weight constraint produced better results overall as illustrated in Figures 3.21b and 3.26d. Moreover, the second best model produced a MSE of 0.009249, very close to the best model and was obtained using the same hyperparameters except the *weight_constraint*, which was set to 2. For that reason, it was decided to use weight constraint in later experiments.

Then, *l2* weight regularization and the optimal value for *weight_constraint* were explored in a third Grid Search using a greater number of epochs. 16 iterations were run with the values listed in Table 3.33

In this Grid Search, models using *l2* weight regularization performed better than those without it as illustrated in Figure 3.27a. However, results from previous Grid Search without *l2* weight regularization were not overcome. Probably, if *l2* weight regularization is not implemented the error increases after some epochs due to overfitting. That could be the reason why models with *l2* 0 performed better when trained for 50 epochs instead of 100.

The best value for *weight_constraint* was 1, closely followed by 4 as illustrated in Figure 3.27b. Therefore, *l2* 0, *weight_constraint* 1 and *epochs* 50 were selected for future searches.

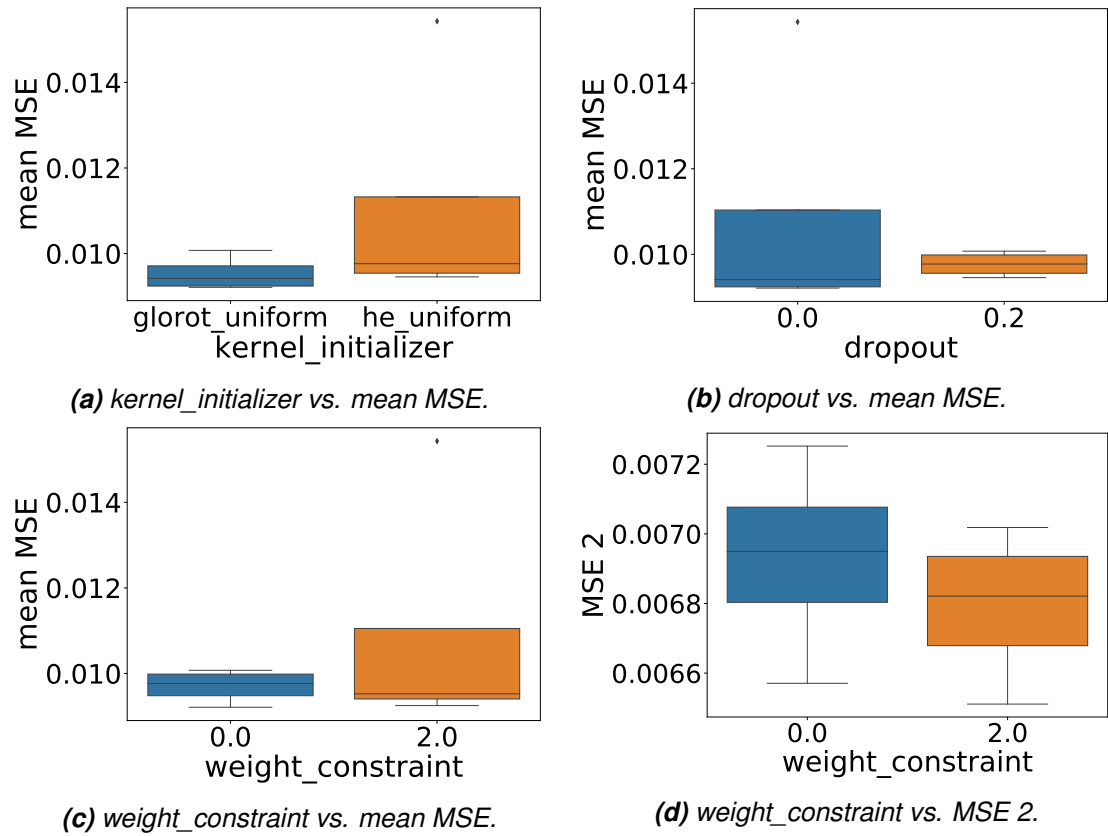


Figure 3.26. Second search results.

Table 3.33. Third search hyperparameter values. Grid Search with 16 iterations.

Identifier	Hyperparameter	Range values	Best found
1	<i>neurons_1</i>	25	25
2	<i>neurons_2</i>	25	25
3	<i>third_layer</i>	False	False
4	<i>neurons_3</i>	—	—
5	<i>output_activation</i>	'linear'	'linear'
6	<i>hidden_activation</i>	'relu'	'relu'
7	<i>optimizer</i>	Adam	Adam
8	<i>lr</i>	0.001	0.001
9	<i>momentum</i>	—	—
10	<i>nesterov</i>	—	—
11	<i>epochs</i>	100	100
12	<i>batch_size</i>	128	128
13	<i>dropout</i>	0	0
14	<i>l2</i>	0, 10^{-6} , 10^{-5} , 10^{-4}	10^{-4}
15	<i>weight_constraint</i>	1, 2, 3, 4	4
16	<i>kernel_initializer</i>	'glorot_uniform'	'glorot_uniform'
17	input sequence length	24	24
mean MSE			0.009439

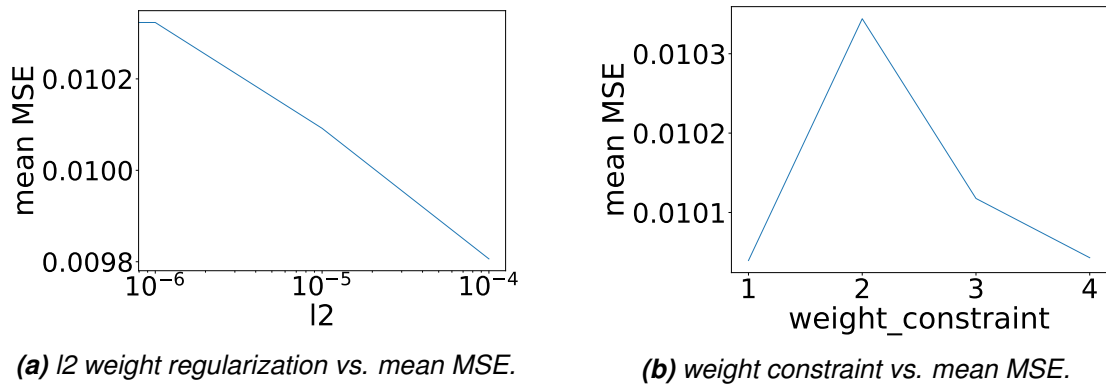


Figure 3.27. Third search results.

Then, the optimal number of neurons in the first and second hidden layers was studied through a Random Search of 15 iterations. The values explored are listed in Table 3.34.

Table 3.34. Fourth search hyperparameter values. Random Search with 15 iterations.

Identifier	Hyperparameter	Range values	Best found
1	<i>neurons_1</i>	20, 30, 40, ..., 20	20
2	<i>neurons_2</i>	20, 30, 40, ..., 50	50
3	<i>third_layer</i>	False	False
4	<i>neurons_3</i>	—	—
5	<i>output_activation</i>	'linear'	'linear'
6	<i>hidden_activation</i>	'relu'	'relu'
7	<i>optimizer</i>	Adam	Adam
8	<i>lr</i>	0.001	0.001
9	<i>momentum</i>	—	—
10	<i>nesterov</i>	—	—
11	<i>epochs</i>	50	50
12	<i>batch_size</i>	128	128
13	<i>dropout</i>	0	0
14	<i>l2</i>	0	0
15	<i>weight_constraint</i>	1	1
16	<i>kernel_initializer</i>	'glorot_uniform'	'glorot_uniform'
17	input sequence length	24	24
mean MSE			0.009448

The best results were achieved by configurations with fewer number of neurons as illustrated in Figure 3.28, which shows the mean validation error obtained by each configuration.

Therefore a second Random search of 5 iterations was run limiting the number of neurons in the first and second hidden layers to 35 and 65 respectively as shown in Table 3.35.

As thought, decreasing the number of neurons produced better results. Figure 3.29

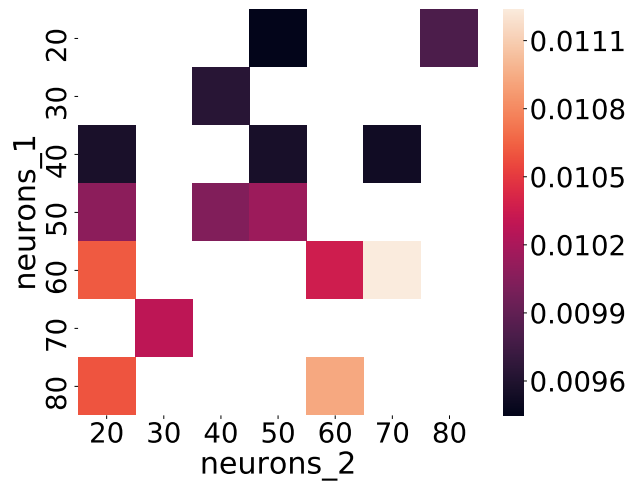


Figure 3.28. Mean validation error obtained by each structure configuration explored in fourth search.

Table 3.35. Fifth search hyperparameter values. Random Search with 5 iterations.

Identifier	Hyperparameter	Range values	Best found
1	<i>neurons_1</i>	10, 15, 20, ..., 35	30
2	<i>neurons_2</i>	10, 15, 20, ..., 65	15
3	<i>third_layer</i>	False	False
4	<i>neurons_3</i>	—	—
5	<i>output_activation</i>	'linear'	'linear'
6	<i>hidden_activation</i>	'relu'	'relu'
7	<i>optimizer</i>	Adam	Adam
8	<i>lr</i>	0.001	0.001
9	<i>momentum</i>	—	—
10	<i>nesterov</i>	—	—
11	<i>epochs</i>	50	50
12	<i>batch_size</i>	128	128
13	<i>dropout</i>	0	0
14	<i>l2</i>	0	0
15	<i>weight_constraint</i>	1	1
16	<i>kernel_initializer</i>	'glorot_uniform'	'glorot_uniform'
17	input sequence length	24	24
mean MSE			0.009194

shows the mean validation error for configurations explored in the fourth and fifth searches. The best model produced 0.009194 mean MSE with a structure of 15 and 30 neurons in the first and second layer respectively. However, a configuration with 30 and 40 neurons, which produced 0.009373 mean MSE, was chosen instead. The reason is that this configuration is located in the area of models with lowest validation error while the configuration with 30 and 15 neurons is located in the border. Therefore, it seems more probable that the configuration with 30 and 40 neurons would keep the good results if the experiments were reproduced.

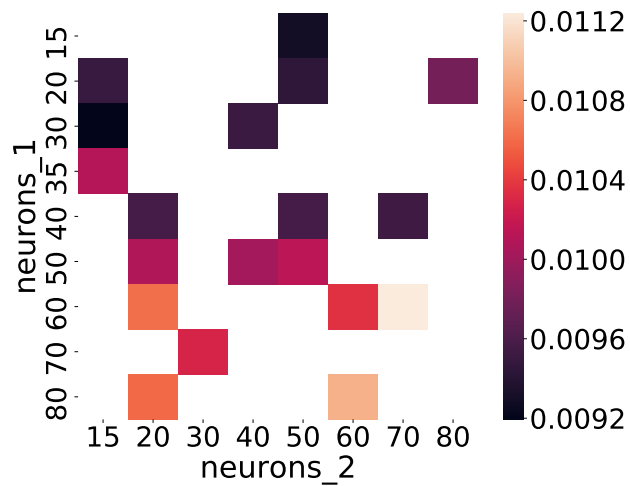


Figure 3.29. Mean validation error obtained by each structure configuration explored in fourth and fifth searches.

At this point, it seems that tuning further the number of neurons might not produce great improvements. Instead, other hyperparameters could be explored but the time required with the available resources would be excessive. Therefore, as the current results are satisfying, the optimization process can conclude. A summary of the whole process is listed in Table 3.36.

Table 3.36. Summary of LSTM hyperparameter optimization process for day ahead forecast.

Model	mean MSE	Improvement (%)
Base model	0.010316	—
First search	0.009449	8.40
Second search	0.009211	10.71
Third search	0.009439	8.50
Fourth search	0.009448	8.41
Fifth search	0.009194	10.88
Final	0.009373	9.14

Then, the final model was trained using the selected configuration and the early stopping technique. Finally, the weights from the best epoch were loaded and the validation error was calculated.

4 RESULTS AND DISCUSSION

The model that performed better forecasting the next hour solar irradiance was the LSTM despite its mean MSE during hyperparameter optimization was slightly lower than Random Forest. This error reduction can be due to increasing the training data in the final training and the use of the early stopping technique which was not implemented during the hyperparameter optimization. Thus, FNNs and LSTMs could train for as many epochs as they needed until the validation error stopped decreasing. The validation RMSE and MAE produced by the different algorithms are listed in Table 4.1.

Table 4.1. Next hour forecast prediction error.

Validation Error	Persistence	RF	FNN	LSTM
RMSE ($W/W/m^2$)	111.50	45.94	44.89	44.42
MAE ($W/W/m^2$)	49.53	21.27	21.12	20.25

The prediction error was calculated using all-time data including nighttime and winter. During the nighttime, the measured solar irradiance and the prediction error are close to zero. Moreover, during winter the magnitude of the measured solar irradiance is considerably lower than during summer. Therefore, the prediction error during daytime and summer is greater than the given values.

To obtain a more realistic measurement of the prediction error, this was calculated again but only for days from April to September and during daylight. The time intervals selected are listed in Table 4.2.

Table 4.2. Conditions of the samples selected from the validation set as true solar time data.

Month	Beginning of daylight [hour]	End of daylight [hour]
April	6	18
May	5	20
June	5	20
July	5	20
August	6	19
September	7	17

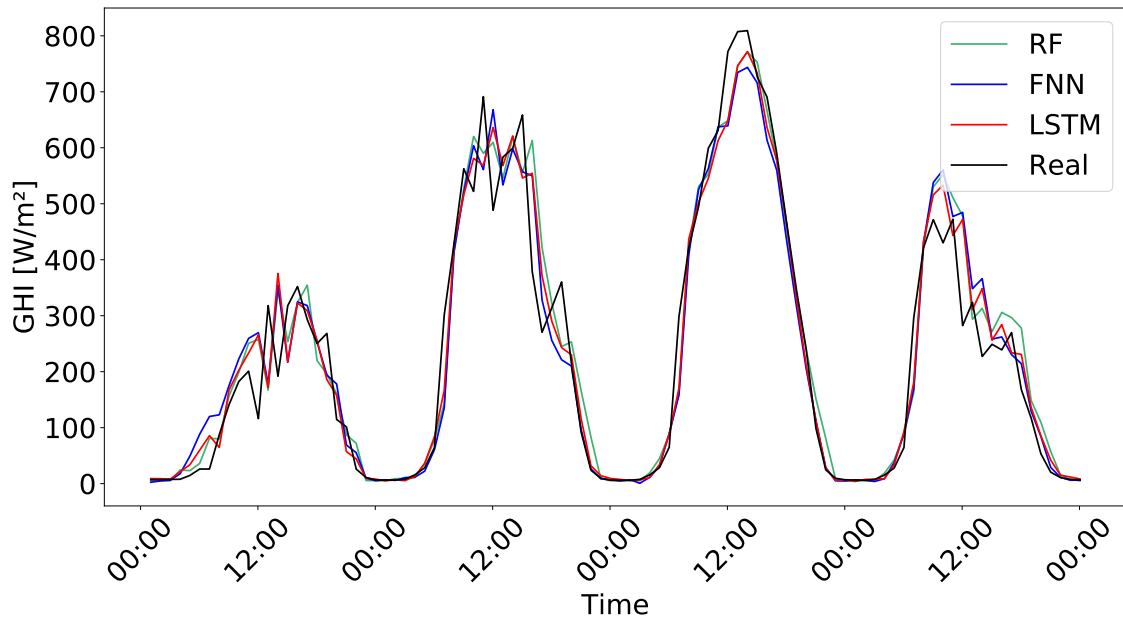
Thus, the prediction errors calculated for the time intervals mentioned above are listed in

Table 4.3.

Table 4.3. Next hour forecast prediction error during true solar time.

Validation Error	Persistence	RF	FNN	LSTM
RMSE (W/m^2)	189.38	77.77	76.98	76.14
MAE (W/m^2)	129.39	52.40	52.86	51.36

The LSTM was expected to considerably outperform the rest of the algorithms due to its complexity. However, LSTM was only 2.1% more accurate than Random Forest and 1.1% than FNN in terms of RMSE. Therefore, the election of the algorithm is left to the time available. If the user has enough time, LSTM is the best option as it offers the highest accuracy. However, if the time is very limited, RF can achieve similar results with little hyperparameter optimization. Figure 4.1 shows a next hour forecast example of 4 days of June. In this example, the real value and the predictions of the 3 machine learning algorithms are illustrated.

**Figure 4.1.** Next hour forecast of 4 days of June.

Generally, all models successfully predicted the next hour solar irradiance on sunny days. Figure 4.2 illustrates the common behavior of the 3 algorithms when predicting solar irradiance on a sunny day. Data from one day of May was selected as an example.

However, the real challenge was to predict the solar irradiance on partly cloudy days. The variability in solar irradiance due to the clouds was difficult to track for the algorithms. Figure 4.3 illustrates a common behavior of the 3 algorithms when predicting solar irradiance on a partly cloudy day. Data from one day of June was selected as an example.

As can be seen, the algorithms normally follow the solar irradiance shape but with 1-hour delay. They might not expect a cloud coming and would forecast a value considerably higher than the real. Then, when they already have data about the cloud one hour later,

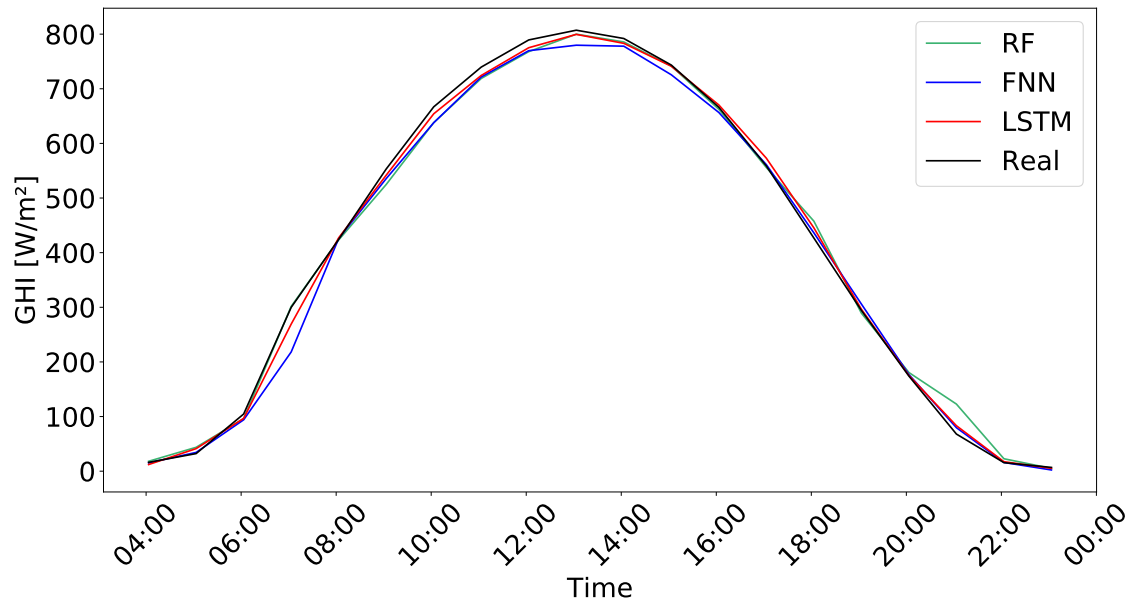


Figure 4.2. Next hour forecast of a sunny day of May.

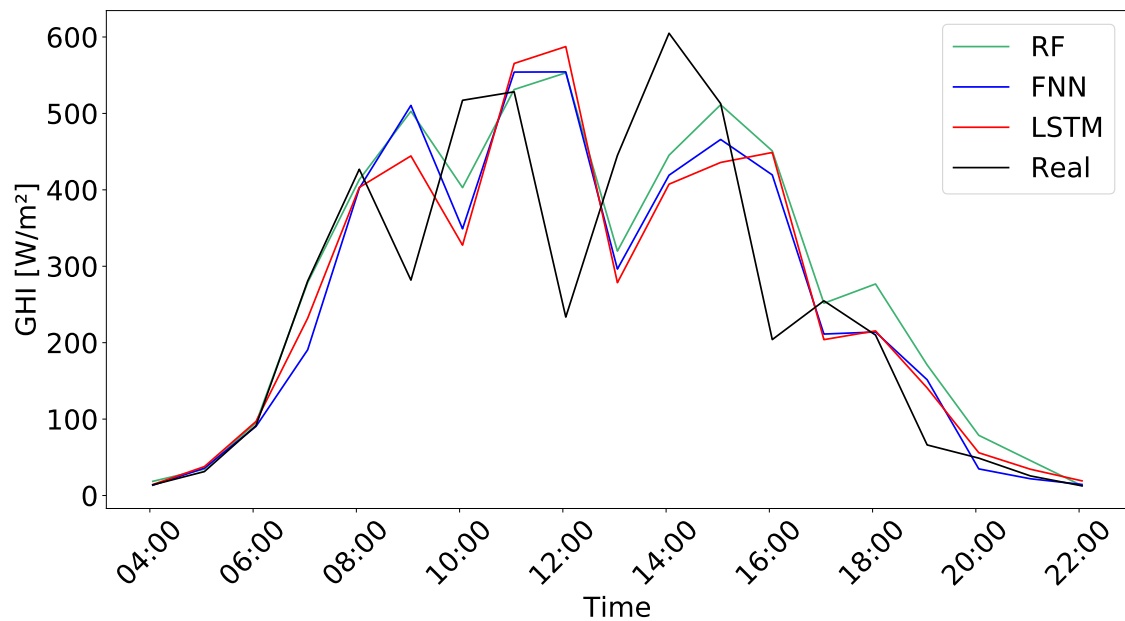


Figure 4.3. Next hour forecast of a partly cloudy day of June.

they use that information to reduce the value of the solar irradiance. Therefore, the major drawback of these approaches is that the algorithms cannot successfully detect the clouds coming or leaving before they do.

This might be solved by importing external data from points near to the weather station such as public historical weather forecast data. Thus, the predictions of some weather variables such as cloud cover could be added for the desired forecast horizons to anticipate sudden changes in cloudiness. Therefore, not only historical data could be used to forecast future solar irradiance but also future weather forecasts. Another proposal is to use a low-cost fish-eye camera for sky images acquisition and use pattern recognition to

track the clouds movement [1].

For the day-ahead forecast, the best model depends on the forecast horizon. The prediction errors for each algorithm are listed in Table 4.4. The prediction error values were calculated for true solar time data with the conditions in Table 4.2.

Table 4.4. Day-ahead prediction error during true solar time.

Validation error	RMSE [W/m^2]				MAE [W/m^2]			
Hours ahead	Persistence	RF	FNN	LSTM	Persistence	RF	FNN	LSTM
1	187.18	84.80	79.67	81.81	128.33	64.38	57.12	58.51
2	187.18	101.90	99.14	99.22	128.33	78.64	73.94	72.51
3	187.18	113.71	111.70	112.32	128.33	88.47	85.31	84.43
4	187.18	122.88	120.65	121.79	128.33	96.26	93.03	92.57
5	187.18	130.22	127.84	129.78	128.33	102.40	99.30	99.36
6	187.18	136.06	132.96	136.10	128.33	107.01	103.56	104.71
7	187.18	140.92	137.11	141.77	128.33	111.16	107.13	109.21
8	187.18	144.94	140.46	146.65	128.33	114.92	110.32	113.81
9	187.18	147.38	142.63	150.05	128.33	116.98	112.26	117.05
10	187.18	148.83	144.06	151.71	128.33	118.20	113.72	118.46
11	187.18	149.49	144.78	152.18	128.33	119.11	114.54	119.03
12	187.18	149.54	145.14	152.79	128.33	119.17	115.02	119.78
13	187.18	149.38	145.37	152.09	128.33	119.15	115.27	119.18
14	187.18	149.18	145.55	151.41	128.33	119.07	115.37	118.67
15	187.18	149.16	145.63	150.09	128.33	119.14	115.40	117.66
16	187.18	149.34	145.69	149.26	128.33	119.40	115.31	117.04
17	187.18	149.45	145.79	148.49	128.33	119.31	115.41	116.47
18	187.18	149.41	145.90	147.95	128.33	119.16	115.60	116.13
19	187.18	149.61	146.01	147.59	128.33	119.43	115.74	115.92
20	187.18	149.87	146.19	147.51	128.33	119.65	116.00	116.12
21	187.18	150.19	146.23	147.14	128.33	119.84	115.98	116.19
22	187.18	150.56	146.29	146.89	128.33	120.17	116.08	115.87
23	187.18	151.28	146.56	147.45	128.33	120.92	116.22	116.56
24	187.18	151.93	146.96	147.63	128.33	121.38	116.61	116.83
Average	187.18	140.42	136.60	139.99	128.33	111.39	107.26	108.84

Moreover, the prediction RMSE and MAE are illustrated in Figure 4.4 and Figure 4.5 respectively. As it can be seen, FNN generally produced the best results despite it was overcome by LSTM in few forecast horizons. LSTM performed properly in the short-term but the prediction error raised considerably for forecasts further than 6 hours unlike FNN. Overall, the prediction error becomes higher as the forecast horizon increases. Thus,

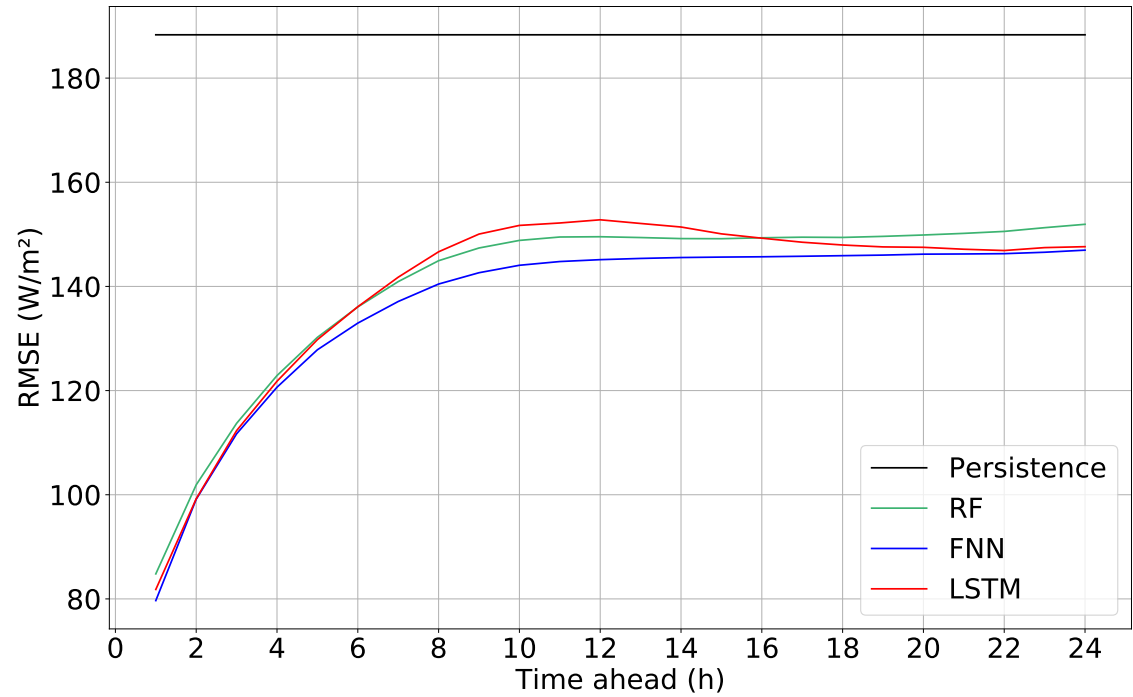


Figure 4.4. Day-ahead forecast prediction RMSE for each algorithm.

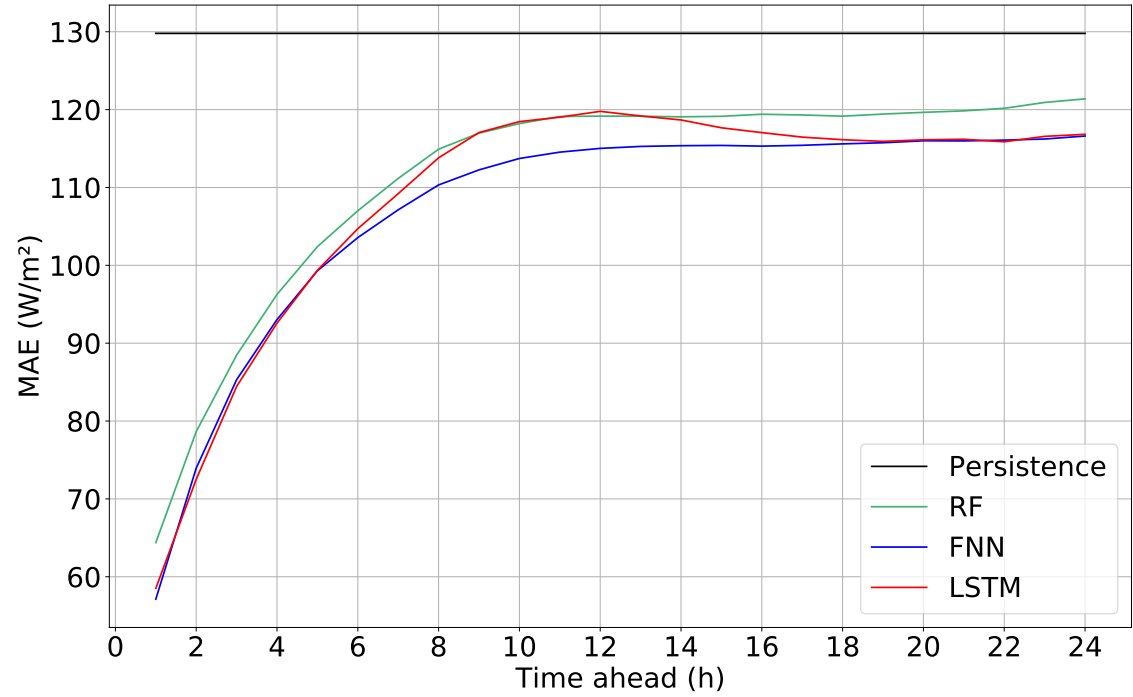


Figure 4.5. Day-ahead forecast prediction MAE for each algorithm.

after 5 hours, the MAE becomes greater than 100 W/m^2 and stabilizes at 10 hours approximately. In that sense, RF and FNN were more stable than LSTM, which had some fluctuations in the long-term. After 10 hours, the error made by machine learning algorithms keeps slowly increasing but is always maintained below the error made by the persistence algorithm.

Besides, when comparing the next hour prediction error values of Tables 4.1 and 4.4, the conclusion is that next hour forecast models predict it considerably better than day-ahead forecast models. Therefore, in terms of RMSE, the error reduction is 8.3% for RF, 3.4% for FNN and 6.9% for LSTM while in terms of MAE the reduction is 18.6% for RF, 7.5% for FNN and 12.2% for LSTM. However, this behavior was expected as one model is calculating only one prediction and the other is calculating 24 at once.

Considering this comparison, a solution that might increase the day-ahead forecast accuracy is to build 24 different models, one for each prediction instead of one for all the predictions. However, that would also require more work that could be used to improve the single model.

Two examples of day-ahead forecast for sunny days are illustrated in Figures 4.6 and 4.7. As shown, the algorithms forecasted a reduced shape of a sunny day without reaching the maximum peak of solar irradiance.

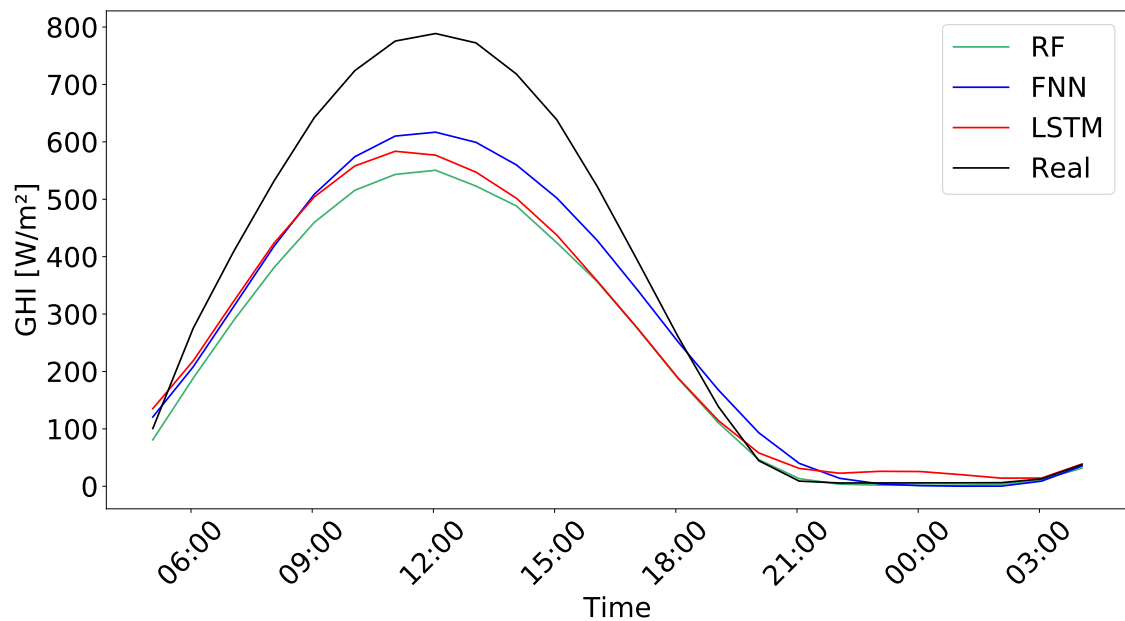


Figure 4.6. Day-ahead forecast of a sunny day of May.

On the other hand, Figures 4.8 and 4.9 illustrate two examples day-ahead forecast for partly-cloudy days. The machine learning algorithms seem to produce a similar approach to the one done for sunny days, forecasting a reduced shape of a sunny day. It seems that the algorithms cannot successfully predict changes in solar irradiance after some hours and forecast then a series of values close to average.

Thus, on both sunny and cloudy days, the algorithms might be avoiding high risks such

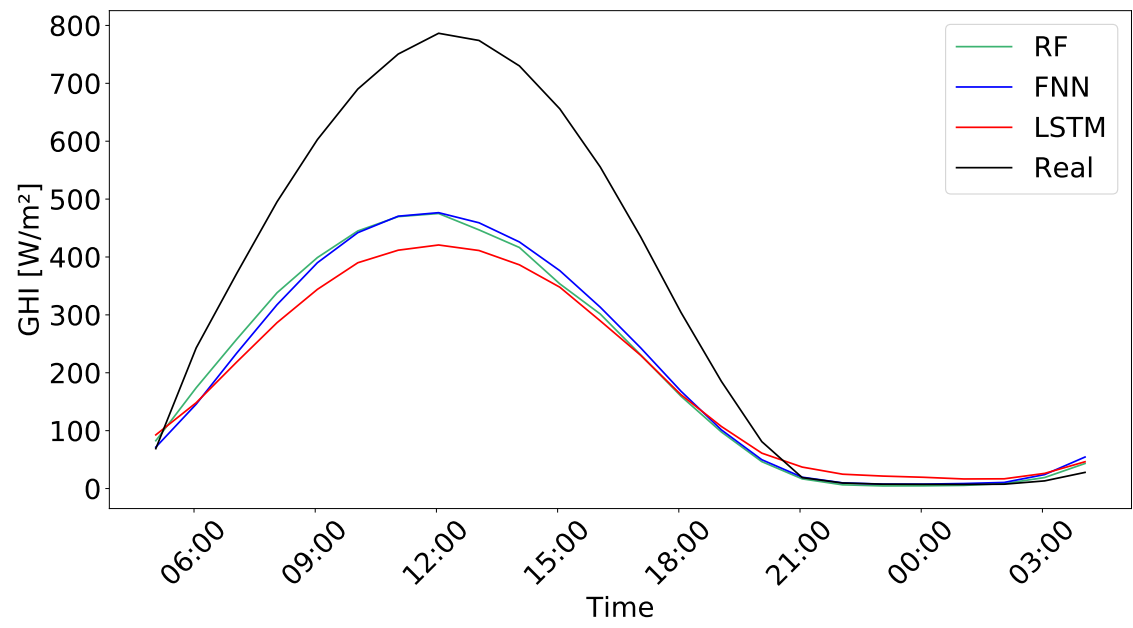


Figure 4.7. Day-ahead forecast of a sunny day of July.

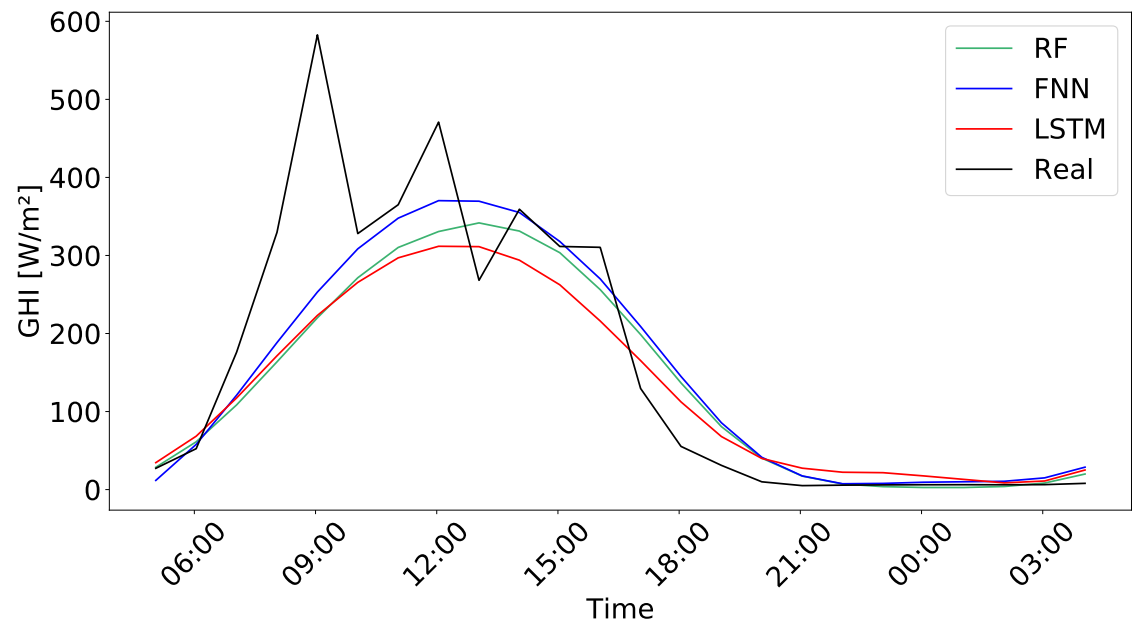


Figure 4.8. Day-ahead forecast of a partly-cloudy day of May.

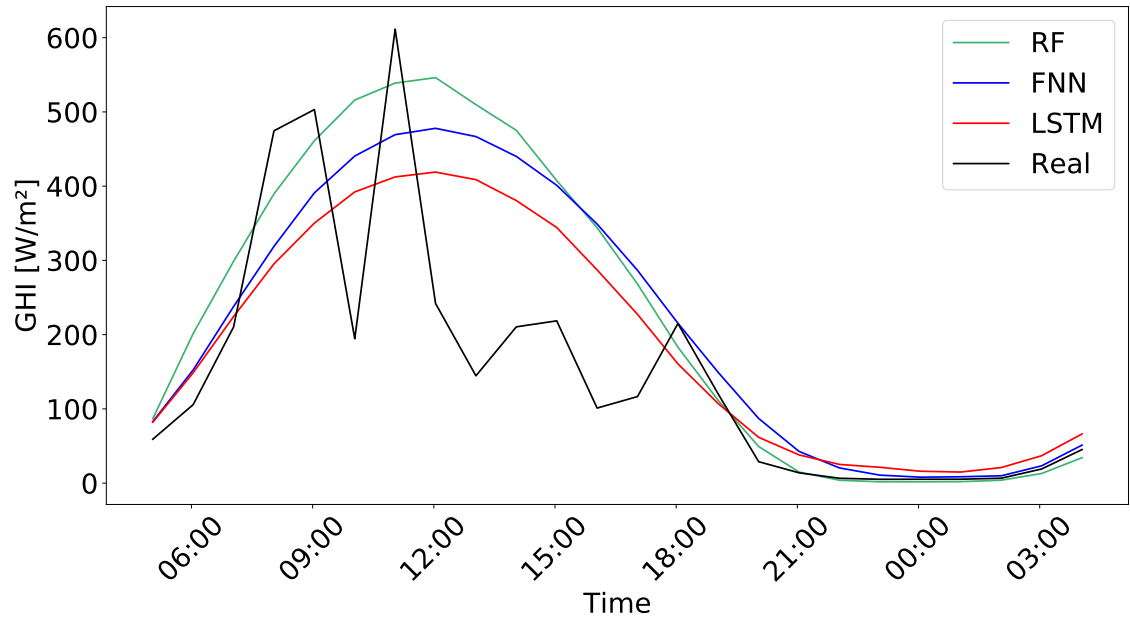


Figure 4.9. Day-ahead forecast of a partly-cloudy day of June.

as forecasting a completely sunny day because if they were wrong the error would be then considerably high. However, the magnitude of the forecasted shape is different for different weather types although it seems to be highly influenced by the cloudiness of the previous 24 hours. Therefore, if a sunny day follows a cloudy day, the algorithms usually forecast low solar irradiance values as shown in Figure 4.7. On the other hand, the example day in Figure 4.6 was preceded by a mostly sunny day and forecasted higher solar irradiance values.

5 CONCLUSIONS

In this study, machine learning algorithms, especially Artificial Neural Networks, have proven to be a great tool to solve time series prediction problems such as solar irradiance forecasting. FNN and LSTM produced the best results although RF achieved also satisfactory results with less time spent optimizing the algorithm.

The algorithms were able to successfully forecast the next hour solar irradiance, especially on sunny days. However, sudden changes in solar irradiance due to cloud movement were difficult to anticipate by the algorithms, even by LSTM, which produced the best results of this forecast.

In day-ahead forecasts, the prediction was satisfactory for short forecast horizons. However further than 5 hours, the MAE was greater than 100 W/m^2 . Again, cloud movement was difficult to forecast. FNN produced the best results of this forecast despite LSTM was expected to be the best predictor as it is widely used in time series forecasting. However, as FNN require less time to train than LSTM, more experiments could be run to optimize its hyperparameters. Therefore, LSTM might overcome FNN with further hyperparameter optimization.

Moreover, next hour forecast models produced better results forecasting the next hour solar irradiance than day-ahead models. Therefore, models that just predict one forecast horizon might perform better than those that make several predictions at once. Therefore, creating a single model for each forecast horizon might increase accuracy.

However, the accuracy is also limited by the data and features fed to the models. Thus, it is important to explore different sources that could bring useful information to forecast solar irradiance.

As said above, the main problem when forecasting solar irradiance was the variability due to clouds. Some recommendations to solve this problem are explained in Chapter 4. For example, importing external data such as public historical weather forecasts from points near to the weather station. Thus, the predictions of some weather variables such as cloud cover could be added for the desired forecast horizons to anticipate changes in cloudiness. Also, adding humidity, wind speed, and wind direction data might improve the prediction accuracy despite they were discarded in this study. Another solution for short-term forecasts is to use pattern recognition on sky images made periodically by a camera placed in the study place to forecast the movement of the clouds and sun. However, this last solution could not be used along with previous historical data.

REFERENCES

- [1] Y. Ai, Y. Peng and W. Wei. A Model of Very Short-term Solar Irradiance Forecasting Based on Low-cost Sky Images. *AIP Conference Proceedings* 1839 (2017).
- [2] A. Alzahrana, P. Shamsi, C. Dagli and M. Ferdowsi. Solar Irradiance Forecasting Using Deep Neural Networks. *Procedia* 114 (2017), 304–313.
- [3] J. Bergstra and Y. Bengio. Random Search for Hyper-Parameter Optimization. *Journal of Machine Learning Research* 13 (2012), 281–305.
- [4] S. Bouktif, A. Fiaz, A. Ouni and M. Adel Serhani. Optimal Deep Learning LSTM Model for Electric Load Forecasting using Feature Selection and Genetic Algorithm: Comparison with Machine Learning Approaches. *Energies* 11.7 (2018).
- [5] P. Bruneau, P. Pinheiro and Y. Didry. Data-driven forecasting of solar irradiance. *Conférence Extraction et Gestion de Connaissances 2018* (2018).
- [6] *Decision Tree Regressor explained in depth*. May 23, 2019. URL: <https://gdcd.com/decision-tree-regressor-explained-in-depth/> (visited on 06/13/2019).
- [7] R. Genuer, J.-M. Poggi and C. Tuleau-Malot. Variable selection using Random Forests. *Pattern Recognition Letters, Elsevier* 31.14 (2010), 2225–2236.
- [8] T. Hastie, R. Tibshirani and J. Friedman. *The Elements of Statistical Learning*. Springer, 2008.
- [9] A. Kuzmiakova, G. Colas and A. McKeenhan. Short-term Memory Solar Energy Forecasting at University of Illinois. (2017).
- [10] S. Mishra and P. Palanisamy. Multi-time-horizon Solar Forecasting Using Recurrent Neural Network. *Energy IEEE Energy Conversion Congress and Exposition* (2018).
- [11] *NREL Solar Resource Glossary*. URL: <https://www.nrel.gov/grid/solar-resource/solar-glossary.html#i> (visited on 05/29/2019).
- [12] X. Qing and N. Yugang. Hourly day-ahead solar irradiance prediction using weather forecasts by LSTM. *Energy* 148 (2018), 461–468.
- [13] *Random Forest Regression model explained in depth*. June 4, 2019. URL: <https://towardsdatascience.com/random-forest-regression-model-explained-in-depth-f2cce437c750> (visited on 06/13/2019).
- [14] *Renewables 2018. Power*. Oct. 8, 2018. URL: <https://www.iea.org/renewables/2018/power/> (visited on 08/08/2019).
- [15] D. Torres Lobera, A. Mäki, J. Huusari, K. Lappalainen, T. Suntio and S. Valkealahti. Operation of TUT Solar PV Power Station Research Plant under Partial Shading Caused by Snow and Buildings. *International Journal of Photoenergy* 2013 (2013).
- [16] *Understanding LSTM Networks*. Aug. 27, 2015. URL: <https://colah.github.io/posts/2015-08-Understanding-LSTMs/> (visited on 06/27/2019).

- [17] *Unsupervised Feature Learning and Deep Learning Tutorial. Multi-Layer Neural Network*. URL: <http://ufldl.stanford.edu/tutorial/supervised/MultiLayerNeuralNetworks/> (visited on 06/01/2019).
- [18] S. Wathmini, F. Lakchani, K. Ahilan, V. Ragupathyraj and K. Anantharajah. Long-term Solar irradiance Forecasting Approaches - A Comparative Study. *Conference on Information and Automation for Sustainability (ICIAfS)* (2018).

A VARIABLE IMPORTANCES

Table A.1. Variable importances found for next hour forecast using Random Forest algorithm.

Position	Variable	Value	Position	Variable	Value
1	GHI t-1	0.8545	51	GHI t-7	0.0006
2	GHI t-20	0.0168	52	Temperature t-23	0.0006
3	DHI t-24	0.0108	53	Temperature t-19	0.0006
4	DHI t-1	0.0107	54	Temperature t-18	0.0006
5	GHI t-19	0.0083	55	Temperature t-17	0.0006
6	GHI t-21	0.0059	56	Temperature t-15	0.0006
7	DHI t-19	0.0058	57	Temperature t-13	0.0006
8	GHI t-18	0.0058	58	Temperature t-10	0.0006
9	DHI t-21	0.0051	59	Temperature t-9	0.0006
10	DHI t-20	0.0043	60	Temperature t-8	0.0006
11	GHI t-17	0.0035	61	Temperature t-5	0.0006
12	d_{cos}	0.0035	62	Temperature t-4	0.0006
13	DHI t-22	0.0025	63	Temperature t-3	0.0006
14	GHI t-2	0.0022	64	DHI t-13	0.0005
15	DHI t-2	0.0019	65	Temperature t-22	0.0005
16	GHI t-22	0.0019	66	Temperature t-21	0.0005
17	DHI t-23	0.0018	67	Temperature t-20	0.0005
18	GHI t-16	0.0017	68	Temperature t-16	0.0005
19	DHI t-18	0.0016	69	Temperature t-12	0.0005
20	GHI t-3	0.0016	70	Temperature t-11	0.0005
21	GHI t-24	0.0015	71	Temperature t-7	0.0005
22	DHI t-16	0.0014	72	Temperature t-6	0.0005
23	GHI t-23	0.0014	73	DHI t-12	0.0004
24	DHI t-3	0.0013	74	DHI t-11	0.0004

Table A.1 continued from previous page

Position	Variable	Value	Position	Variable	Value
25	DHI t-17	0.0012	75	DHI t-10	0.0004
26	DHI t-4	0.0012	76	Humidity t-24	0.0004
27	GHI t-15	0.0012	77	Humidity t-23	0.0004
28	d_{sin}	0.0012	78	Humidity t-21	0.0004
29	GHI t-4	0.0011	79	Humidity t-8	0.0004
30	t_{sin}	0.0011	80	Humidity t-6	0.0004
31	DHI t-7	0.0010	81	Humidity t-4	0.0004
32	DHI t-6	0.0010	82	Humidity t-2	0.0004
33	DHI t-5	0.0010	83	Humidity t-22	0.0003
34	Temperature t-1	0.0010	84	Humidity t-20	0.0003
35	GHI t-14	0.0009	85	Humidity t-19	0.0003
36	GHI t-5	0.0009	86	Humidity t-18	0.0003
37	Temperature t-24	0.0009	87	Humidity t-17	0.0003
38	DHI t-9	0.0008	88	Humidity t-15	0.0003
39	DHI t-8	0.0008	89	Humidity t-14	0.0003
40	GHI t-13	0.0008	90	Humidity t-13	0.0003
41	DHI t-15	0.0007	91	Humidity t-12	0.0003
42	GHI t-12	0.0007	92	Humidity t-11	0.0003
43	GHI t-9	0.0007	93	Humidity t-10	0.0003
44	GHI t-8	0.0007	94	Humidity t-9	0.0003
45	GHI t-6	0.0007	95	Humidity t-7	0.0003
46	Temperature t-14	0.0007	96	Humidity t-5	0.0003
47	Temperature t-2	0.0007	97	Humidity t-3	0.0003
48	DHI t-14	0.0006	98	Humidity t-1	0.0003
49	GHI t-11	0.0006	99	t_{cos}	0.0003
50	GHI t-10	0.0006	100	Humidity t-16	0.0002

Table A.2. Variable importances found for day ahead forecast using Random Forest algorithm.

Position	Variable	Value	Position	Variable	Value
1	GHI t-1	0.2011	51	DHI t-3	0.0036
2	t_{sin}	0.1833	52	GHI t-24	0.0036
3	d_{cos}	0.1690	53	DHI t-2	0.0035
4	t_{cos}	0.0444	54	Temperature t-24	0.0027
5	GHI t-14	0.0264	55	Temperature t-2	0.0025
6	DHI t-13	0.0227	56	Temperature t-4	0.0022
7	GHI t-10	0.0110	57	Temperature t-3	0.0022
8	GHI t-16	0.0106	58	Humidity t-1	0.0022
9	GHI t-15	0.0104	59	Humidity t-24	0.0020
10	DHI t-16	0.0103	60	Temperature t-23	0.0019
11	DHI t-12	0.0091	61	Temperature t-18	0.0019
12	d_{sin}	0.0090	62	Temperature t-13	0.0019
13	DHI t-8	0.0077	63	Temperature t-12	0.0019
14	GHI t-4	0.0074	64	Temperature t-11	0.0019
15	DHI t-1	0.0072	65	Temperature t-9	0.0019
16	GHI t-9	0.0071	66	Temperature t-8	0.0019
17	DHI t-9	0.0069	67	Temperature t-7	0.0019
18	DHI t-18	0.0068	68	Temperature t-6	0.0019
19	GHI t-13	0.0068	69	Temperature t-5	0.0019
20	GHI t-8	0.0068	70	Temperature t-22	0.0018
21	GHI t-3	0.0067	71	Temperature t-21	0.0018
22	GHI t-17	0.0065	72	Temperature t-20	0.0018
23	GHI t-11	0.0063	73	Temperature t-19	0.0018
24	GHI t-12	0.0062	74	Temperature t-17	0.0018
25	GHI t-2	0.0062	75	Temperature t-16	0.0018
26	DHI t-14	0.0061	76	Temperature t-15	0.0018
27	GHI t-5	0.0057	77	Temperature t-14	0.0018
28	GHI t-7	0.0055	78	Temperature t-10	0.0018
29	DHI t-5	0.0054	79	Humidity t-23	0.0015
30	DHI t-20	0.0053	80	Humidity t-2	0.0015

Table A.2 continued from previous page

Position	Variable	Value	Position	Variable	Value
31	GHI t-6	0.0052	81	Humidity t-22	0.0014
32	DHI t-11	0.0050	82	Humidity t-20	0.0014
33	GHI t-21	0.0050	83	Humidity t-4	0.0014
34	GHI t-22	0.0049	84	Humidity t-3	0.0014
35	DHI t-15	0.0048	85	Humidity t-21	0.0013
36	DHI t-10	0.0048	86	Humidity t-19	0.0013
37	DHI t-7	0.0047	87	Humidity t-15	0.0013
38	DHI t-17	0.0046	88	Humidity t-14	0.0013
39	DHI t-6	0.0046	89	Humidity t-13	0.0013
40	GHI t-23	0.0043	90	Humidity t-10	0.0013
41	DHI t-22	0.0042	91	Humidity t-9	0.0013
42	GHI t-19	0.0042	92	Humidity t-7	0.0013
43	GHI t-18	0.0042	93	Humidity t-6	0.0013
44	DHI t-23	0.0041	94	Humidity t-5	0.0013
45	DHI t-4	0.0040	95	Humidity t-18	0.0012
46	DHI t-21	0.0039	96	Humidity t-17	0.0012
47	DHI t-19	0.0039	97	Humidity t-16	0.0012
48	GHI t-20	0.0039	98	Humidity t-12	0.0012
49	DHI t-24	0.0037	99	Humidity t-11	0.0012
50	Temperature t-1	0.0037	100	Humidity t-8	0.0012